AFIT/GAP/ENP/96D-09

Modeling and Simulation of Optical Characteristics of Microelectromechanical Mirror Arrays

THESIS

Peter C. Roberts
Captain, USAF

AFIT/GAP/ENP/96D-09

Approved for public release; distribution unlimited

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U. S. Government.

AFIT/GAP/ENP/96D-09

Modeling and Simulation of Optical Characteristics of Microelectromechanical Mirror Arrays

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science

Peter C. Roberts, B.S., M.B.A.
Captain, USAF

17 December, 1996

AFIT/GAP/ENP/96D-09

Modeling and Simulation of Optical Characteristics of Microelectromechanical Mirror Arrays

Peter C. Roberts, B.S., M.B.A.
Captain, USAF

Approved:

Michael C. Roggemann, Major, USAF Chairman, Advisory Committee	Date
Byron M. Welsh Member, Advisory Committee	Date
Victor M. Bright Member, Advisory Committee	Date

Table of Contents

List of Figures

List of Tables

List of Symbols

List of Abbreviations

Abstract

MEMS (Micro-Electro-Mechanical Systems) micromirror devices can be used to control the phase of a propagating light wavefront, and in particular to correct aberrations that may be present in the wavefront, due to either atmospheric turbulence or any other type of fixed or time and space varying aberrations. In order to shorten the design cycle of MEMS micromirror devices, computer software is developed to create, from MEMS micromirror device design data, a numerical model of the MEMS device. The model is then used to compute the far field diffraction pattern of a wavefront reflected from the device, and to predict the effectiveness with which it can be used to correct an aberrated wavefront. For validation, the computed far field diffraction pattern is compared to that measured using a real MEMS micromirror device, with a reasonable match between the two being found. The model is designed for maximum flexibility and can be easily adapted to new designs of MEMS micromirror devices.

Modeling and Simulation of Optical Characteristics of Microelectromechanical Mirror Arrays

I. Introduction

This thesis describes the creation and use of a computer model to predict the optical performance of the MUMPS9 hexagonal micromirror array, AFIT's most recent design of a microelectromechanical deformable mirror (MEM-DM). Most of the effort involved writing software that creates the model based on a given set of data describing the device. The goal was to be able to easily model other devices as well, so that the performance of a proposed design can be predicted (and optimized) before fabricating it. The MUMPS9 model was then used to show that such a device is capable of correcting an aberrated wavefront, as physically demonstrated by Hick (15). The flexibility of the modeling software is demonstrated by creating a model of the next proposed design iteration, and showing that it should perform better than the MUMPS9 device.

This chapter provides the background and theory for the thesis. The first section provides the overall background and statement of the problem. This is followed by three sections that cover the background in more detail: one describes atmospheric turbulence as it applies to the problem of image distortion; the next one covers the basic theory of adaptive optical (AO) systems, a technology used to overcome the effects of atmospheric turbulence; and the third describes the MUMPS9 device, the most recent evolution of a new type of deformable mirror, a key technology of AO systems. Finally, an overview of the remainder of the thesis is provided.

1.1 *Background and Problem Statement*

It is a well-known effect of atmospheric turbulence that light propagating through it becomes distorted. Because of this, the atmosphere places a limit on the resolution of optical imaging devices that is lower than would be theoretically possible. One method of overcoming the effects of turbulence is to somehow measure the amount of distortion in the incoming wavefront, and then, in real time, to correct for it. This is the premise of adaptive optics. One of the key ingredients of an AO system is an optical device that can be controlled precisely and quickly enough to correct an incoming wavefront. AFIT is developing such a device, an array of micromirrors fabricated using microelectromechanical systems (MEMS) technology.

In order to shorten the design cycle of MEM-DMs, it is necessary to simulate the optical characteristics of these devices with a computer model. To be useful, the model must be flexible and easily adaptable to new designs.

1.2 *Atmospheric Turbulence and Image Distortion*

> Turbulence was too dangerous to waste time on. It seemed almost unknowable. There was a story about the quantum theorist Werner Heisenberg, on his deathbed, declaring that he will have two questions for God: why relativity, and why turbulence. Heisenberg says, "I really think He may have an answer to the first question." (9)

From the sun's heating rays to butterflies' fluttering wings, countless factors cause different parts of the atmosphere to have different temperatures, pressures and humidity. The atmosphere is constantly trying to equalize itself, but thanks to the nonlinear relationship between the aforementioned parameters, it does not do so in a very smooth manner. The result is what is called turbulence (9). Unfortunately, the varying qualities of the atmosphere also cause the index of refraction within the atmosphere to vary. This, in combination with larger scale winds and currents, causes the index of refraction experienced by an optical wavefront entering an imaging device to vary rapidly in both space and time across the aperture of the device.

By the time light from a star or a satellite reaches a telescope on the earth's surface, what started as a nearly flat wavefront is seriously aberrated. This can be observed on any clear night as the twinkling of the stars.

The spatial scale of the variation, r_0, was defined by Fried as the "diameter of a collector for which distortion effects begin to seriously limit performance" (5), or more rigourously, to be "the largest aperture within which the total rms wavefront irregularity is less than one radian" (23). Thompson gives typical values for r_0 of 5–15cm (23). Because of turbulence, many optical systems are unable to achieve their theoretical minimum angular resolution of λ/D, where λ is the wavelength of the incoming light and D is the aperture of the optical system. Instead, their resolution is limited to λ/r_0 (6). For a telescope with a diameter of 2m, the difference at $\lambda = 700$nm can be as dramatic as that between a theoretical limit of 0.07″and a turbulence-induced limit of 1″($r_0 = 15$cm) (8).

Atmospheric turbulence is of special concern to astronomers, who have over 20km of atmosphere affecting their view. Newton suggested observing from "the tops of the highest Mountains" (18:111). Apparently, many astronomers have followed his advice, so that "Mount" is a common first name for observatories: Mount Palomar, Mount Wilson, Mount Haleakala, Pic du Midi, Cerro Tololo. Yet even at these high altitudes, turbulence still takes its toll. For the best mountaintops, Thompson claims $r_0 \simeq$ 20–30cm (23). The Hubble Space Telescope is evidence of how far (and to what expense) astronomers are willing to go to bypass the atmosphere.

1.3 Adaptive Optics

In 1953, before putting telescopes in space was technologically feasible, Babcock proposed making an optical system that could measure the amount of distortion caused by the atmosphere, and correct for it in real time (12). This is the premise of adaptive optics. Unfortunately, in 1953, such a system was not technologically feasible either. It was not until more than twenty five years later that the US Air Force

created the first AO system, on top of Mount Haleakala, to take pictures of orbiting satellites. Currently, development of AO systems is occuring worldwide (13).

A basic AO system is depicted in figure 1. A flat incoming wavefront propagates through the atmosphere, and its phase becomes distorted. It is this aberrated wavefront that goes through the telescope's optics. With a traditional nonadaptive system, the aberrated wavefront coming out the other end of the telescope is what would be observed, with the reduced resolution described above. With an adaptive system, the aberrated output of the telescope is first reflected on a flat mirror that tilts to eliminate the overall tilt in the wavefront, keeping the image in a constant location in the image plane. The wavefront is then reflected on a deformable mirror, which has been adjusted to correct the phase distortion. Next, a beamsplitter sends a portion of the corrected wavefront to a wavefront sensor, the output of which is used to calculate the corrective input that must be provided by the deformable mirror and tilt compensator. The portion of the corrected wavefront that does not go to the wavefront sensor is used to form the observed image.

Another design criterion of note is related to the fact that since r_0 defines the scale of the variations in the wavefront, it must also correspond to the scale of the corrections. To perform optimally, the aperture must be broken down into an array of r_0-sized subapertures, and the phase in each one individually measured and corrected. This means that a 1.5m telescope, for example, will need over 200 subapertures, each individually correctable more than 300 times every second (23).

One of the key shortcomings that made AO impractical in the 1950's was the need for a wavefront-correcting device with the required resolution and speed. The fact that AO has been defined as "the technique of using deformable mirrors to compensate in real time for the atmosphere's ever-changing distortion of an image" (1) illustrates how deformable mirrors are the preferred device to correct the wavefront. However, other devices, such as crystals whose index of refraction can be made to vary with an electric field, have also been investigated (13).

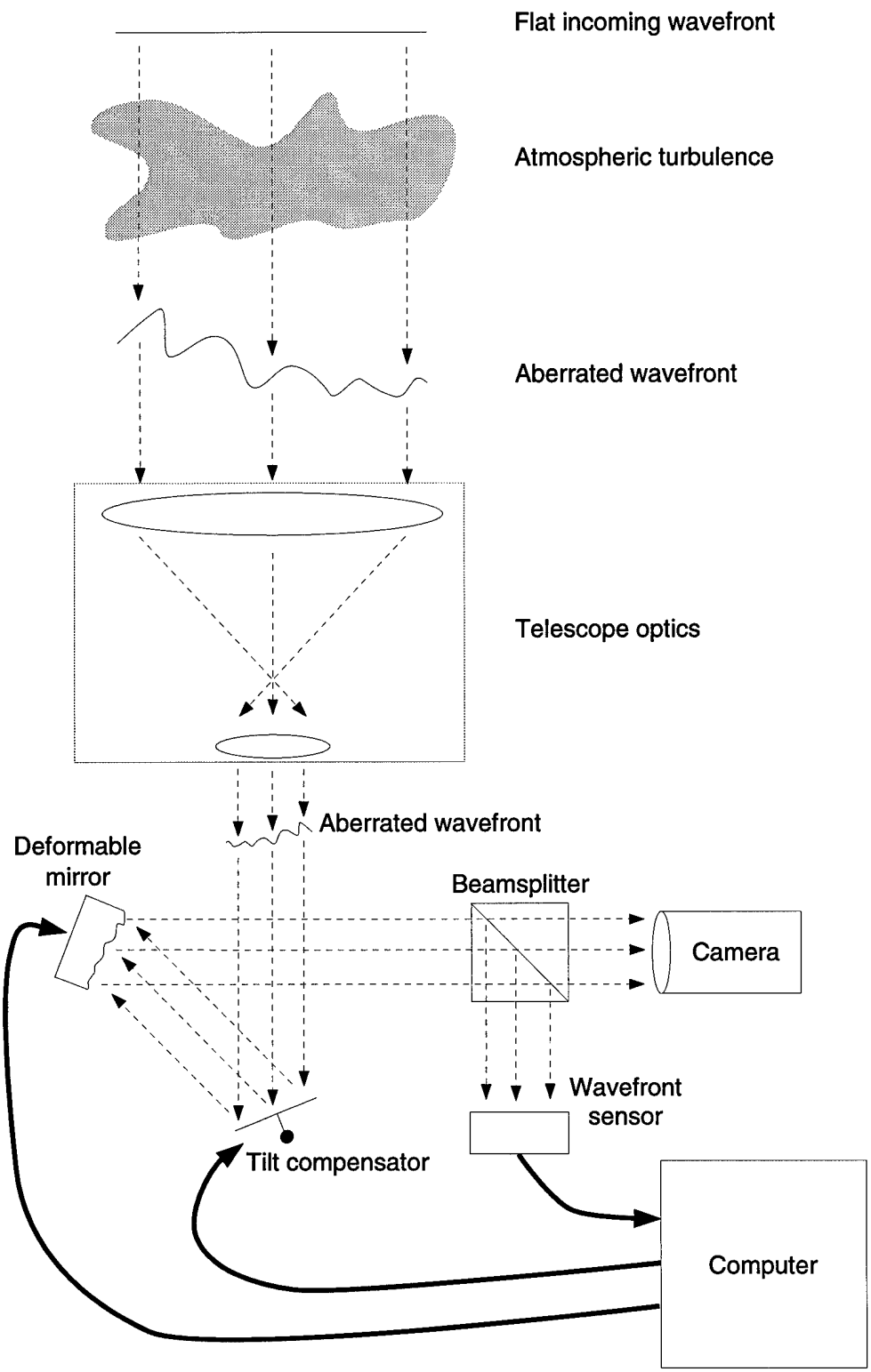

Figure 1. Basic adaptive optical system.

Most current AO systems use a deformable continuous membrane mirror (13), which is rather expensive to produce and maintain. A potentially much cheaper approach involves using an array of MEMS micromirrors constructed using the same fabrication techniques that semiconductor makers have been aggressively developing for decades. Such micromirror systems themselves have actually been under development for over ten years now also, with Texas Instruments nearing commercialization with a high definition (1280 $\times$ 1024 pixels, 60 frames/s) display system (16). However, TI's "digital micromirror device" (DMD) acts as a light switch by deflecting a beam to turn given pixels on or off, rather than as a phase control device to modify a wavefront.

1.4 Microelectromechanical Mirror Arrays

Many different types of MEMS micromirrors have been constructed. Michalicek (17) provides a complete overview of devices. Since the goal of this thesis is to model AFIT developed MEM-DMs, the following description will cover the current, most evolved device: the MUMPS9 hexagonal micromirror array produced by Comtois (2) and shown in Figure 2. An individual micromirror, with the flexures visible, is shown in Figure 3.

MUMPS (Multi-User MEMS Process) fabrication involves the now commonplace method of depositing a layer of material on a silicon substrate, etching part of the design, depositing another layer, and so on. The materials used, the order of deposition, and the thickness are shown in Table 1 (2:2-30).

The "Poly" layers are made of polycrystalline silicon, which provides both the circuitry and the mechanical properties of the device. The oxide layers are removed by an acid bath after all the layers have been deposited. This makes it possible to construct a polysilicon mirror held in place by polysilicon flexures over an empty space (where oxide has been removed), above an addressable electrode made with a lower layer of polysilicon. Creating a voltage potential between the mirror and the

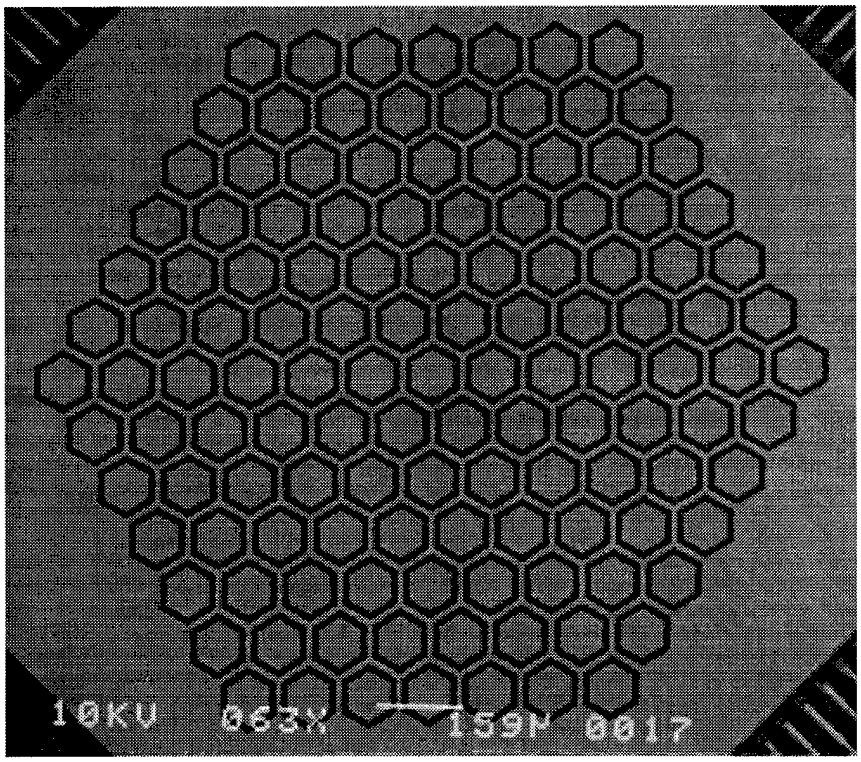

Figure 2. MUMPS9 micromirror array.

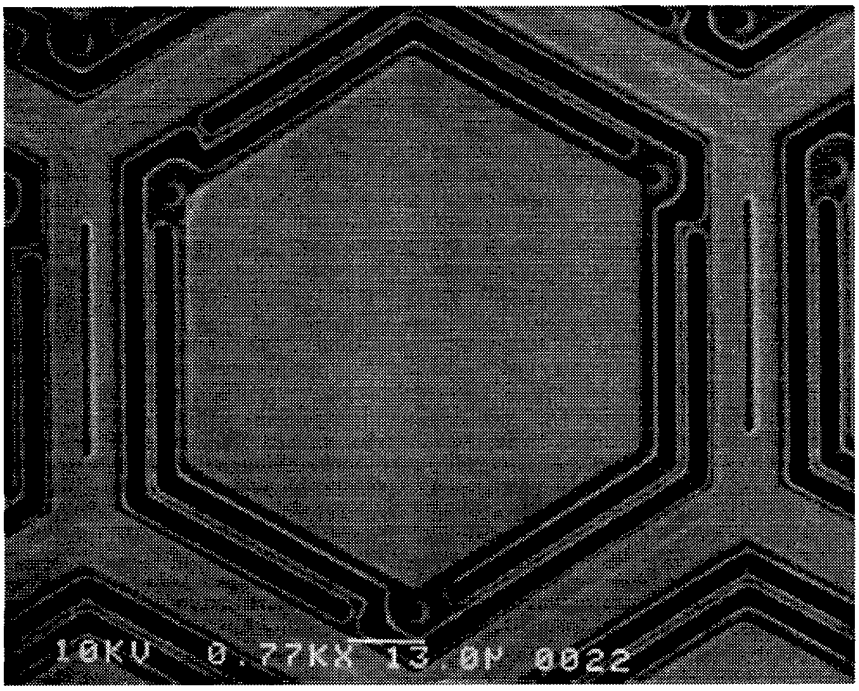

Figure 3. MUMPS9 micromirror showing flexures.

electrode causes the mirror to be pulled downward, with the flexures providing the restoring, upward force. A very precise analysis of the electromechanical functioning of the device was performed by Michalicek (17). The final layer to be deposited is a thin layer of gold, used for its high reflectivity.

For the purpose of modeling the optical performance of the device, both the physical dimensions and reflectivity of the various parts of the device must be known. Referring to Figure 3, the width of an individual mirror, corner-to-corner and including both the gold plated area and the small edge of exposed Poly2, is 100μm. The aforementioned edge, as well as the flexures and the outside edge of exposed Poly2, are 2μm wide. The gaps between the flexures and the edges are 3.1μm wide. Finally, the width of the interval between micromirrors, including the exposed Poly2 edges, is 10μm (2:A-28).

In Figure 3, a narrow depression can be seen along each of the vertical inter-mirror areas. These depressions are where the Poly2 layer is anchored to the Poly1

layer, which is in turn anchored to the nitride layer. They are artifacts of the fabrication process: during fabrication, the Oxide2 layer was etched away here to allow the Poly2 layer to be deposited directly over the Poly1 layer, causing the depression. There are also three small circular depressions that can be seen at the corners of the mirror where it is attached to the flexure arms. These are dimples that were deliberately put in to prevent the mirror from shorting out with its electrode should it be inadvertantly pulled down all the way.

Finally, the reflectivities of the various materials, at the wavelength of a Helium-Neon (HeNe) laser ($\lambda = 632.8$nm), are given in Table 2.

1.5 Overview of the Thesis

The thesis is organized into four chapters and three appendices. The first chapter introduced the reader to the problem and its background. The remaining chapters cover the development and application of the models, and the appendices describe the software used to actually produce them.

The second chapter covers the mathematical methods used to create a model of the MUMPS9 hexagonal micromirror array, and the use of the model to predict the optical performance of that device. The third chapter describes the results obtained from modelling the correction of a spherically aberrated wavefront with a MUMPS9 device and compares them to those obtained with a real device. Conclusions are drawn in the final chapter.

The first two appendices provide a "User Manual" for the modelling software. Appendix A shows how to use the software to create a model, while Appendix B explains how to adapt the software to model a different MEMS design. Appendix C is a listing of all the source code.

Order	Material	Thickness (μm)	Use
7	Gold	0.5	reflective mirror coating
6	Poly2	1.5	flexures, mirror, anchors
5	Oxide 2	0.75	sacrificial layer
4	Poly1	2.0	anchors
3	Oxide 1	2.0	sacrificial layer
2	Poly0	0.5	wires, electrodes
1	Nitride	0.6	substrate
0	Silicon	200	"bedrock"

Table 1. MUMPS layers.

Material	Reflectivity
Gold	0.9
Poly2/1/0	0.35
Nitride	0.4

Table 2. MUMPS reflectivities at $\lambda = 632.8$nm.

II. Methodology

2.1 Overview

The process of modeling the optical characteristics of a MEM-DM involves the following steps:

1. Create a sampled reflectivity function, $R_{m,n}$ representing the device being modeled. Each sampled point has the complex value $R = re^{i2\pi\phi}$, where r represents the absolute reflectivity at that point and ϕ is the phase advance, relative to a reference plane, caused by reflection at that point.
2. Create a sampled wavefront function $W_{m,n}$ that has the same resolution and number of samples as $R_{m,n}$.
3. Define the optical disturbance in the source plane: $U_{m,n} = W_{m,n} \times R_{m,n}$.
4. Using Fourier optics theory, have the computer calculate the normalized far field diffraction intensity pattern $I_{u,v}$.
5. Determine the spatial sampling of the predicted pattern.

The following sections describe in detail how each of the above steps was accomplished while modelling the MUMPS9 hexagonal micromirror array.

2.2 The reflectivity function $R_{m,n}$

To take advantage of the symmetry in an array of micromirrors, the first step in creating the reflectivity function was to sample an individual micromirror. This function was then repeated, or tiled, into a larger array of sampled points representing an array of micromirrors. Two real-valued functions were created: $r_{m,n}$ representing the absolute reflectivity of a sample point, and $h_{m,n}$ representing the height, above a reference plane, at each sample point on the MEMS device. To simulate deflection of individual micromirrors, the height function $h_{m,n}$ was adjusted. Finally, $h_{m,n}$ and

Index	Material	Height (μm)	Reflectivity
1	Nitride	0	0.4
2	Poly2	2.75	0.35
3	Mirror	4.25	0.9
4	Anchor	4.25	0.9

Table 3. Area indices.

$r_{m,n}$ were used to form the complex-valued $R_{m,n}$. One design goal of the model was to have a sample spacing less than half the width of the smallest feature, to satisfy the Nyquist criterion for a bandwidth limited function (3). The smallest features of the MUMPS9 device are the 2μm wide flexure arms. However, since the arms are not lined up with the cartesian pixel grid, but are rotated a minimum of $\pi/12$ radians, the maximum sample spacing that satisfies the Nyquist criterion is 1.04μm. Modelling the MUMPS9 device with 1024×1024 samples produced a $\Delta\xi$ of 1.491μm. Reducing the sample spacing would have required using 2048×2048 samples, which was beyond the capabilities of readily available computers. Since the flexures are also the least reflective elements of the device, it was deemed an acceptable compromise to use the next smallest feature to set the Nyquist limit. The gaps between the flexure arms are 3.1μm wide, which leads to a maximum acceptable sample spacing of 1.60μm. This is satisfied by the 1024×1024 sample model.

2.2.1 Representation of an individual micromirror. The individual micromirror sample function was created using computer graphics techniques. The function was represented as an "image" (actually a two- dimensional array in a computer), with each picture element, or pixel, corresponding to one sample point. The image was defined as a set of four areas with common height and reflectivity, with each area being assigned an index according to table 3. Each of these areas was then broken down into a collection of closed, convex polygons. Each of these was in turn defined by its vertices, and one additional point inside the polygon. Finally, the

coordinates of all the points were converted from continuous coordinates, measured in μm, to pixel coordinates, by setting the size of the finished image to be 90×90 pixels. Low-level graphics algorithms were used to draw the lines surrounding the polygons, and then to set each pixel within to the appropriate area index.

2.2.1.1 Vertices. The first set of information to be defined were the coordinates of all sixty three vertices in the image. These were calculated using the geometric information shown in Figure 4, where the vertices for each of the two kinds of corners are represented by the white dots. The coordinates of these vertices were first calculated in axes convenient to each of the two kinds of corners. They were then rotated three times to make three sets of vertices representing each of the three corners of each kind. The orientation of the image has two reasons. First, it matches the orientation of the actual micromirrors relative to the drawing grid used during fabrication. This orientation ensures that none of the flexures are parallel to the drawing grid, and therefore they all end up the same width (2). Second, it is the most efficient way to pack a hexagon into a square. This orientation therefore provides the smallest sample spacing $\Delta\xi$ for a given size image (measured by the number of pixels).

2.2.1.2 Lines. Next came the simple but tedious process of defining the list of lines. For each line shown in Figure 5, an entry was made in the list, which referred to the list of vertices described above. A MUMPS9 micromirror image consists of one hundred eleven lines, each defined by the indices of its end points (in the list of vertices) and the index (from Table 3) of the area it borders.

2.2.1.3 Seed points. The micromirror image is composed of fortynine polygons. For each of these, a seed point was used to define the interior of the polygon and used to fill it with the appropriate area index value. The seed point

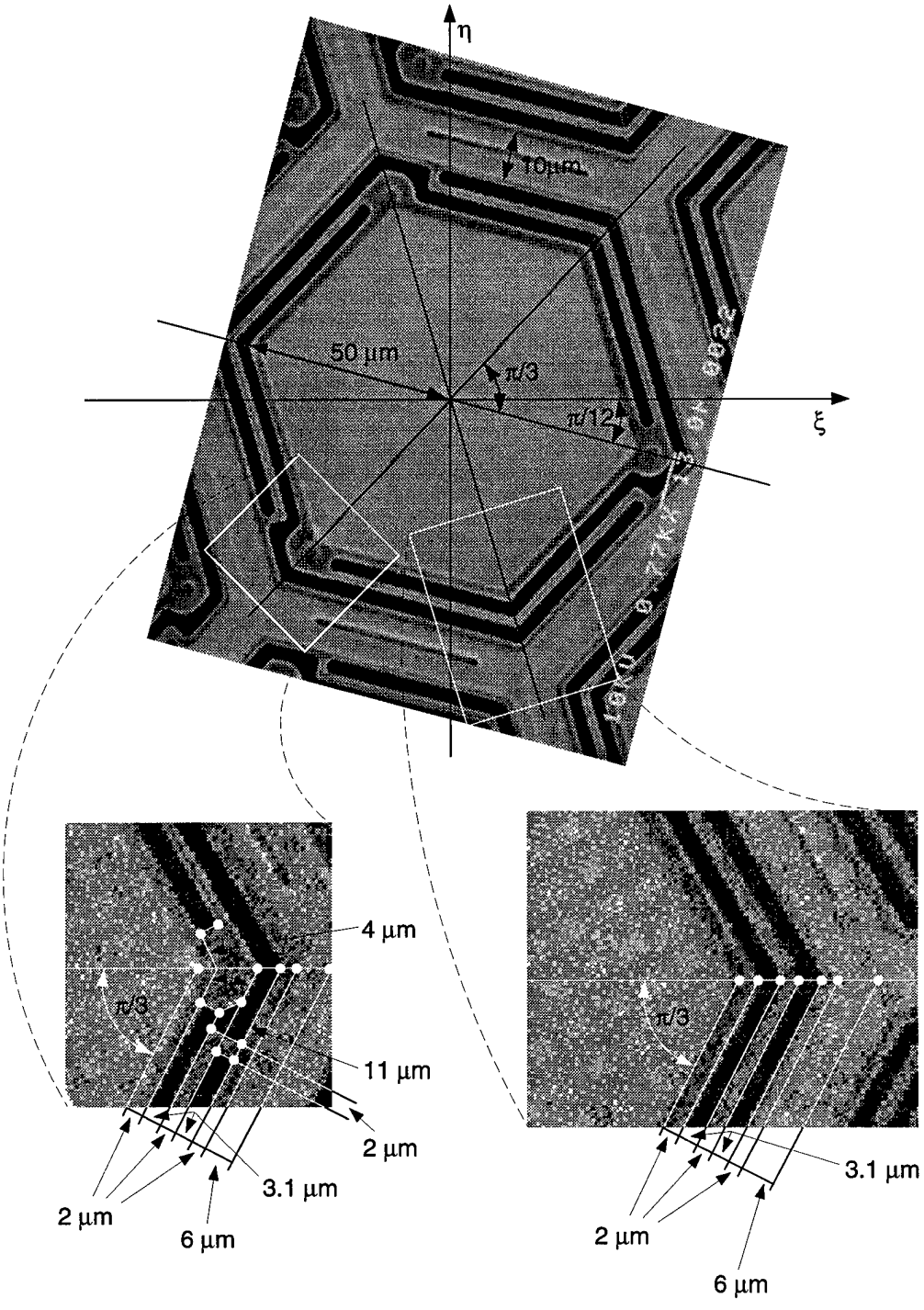

Figure 4. MUMPS9 geometry.

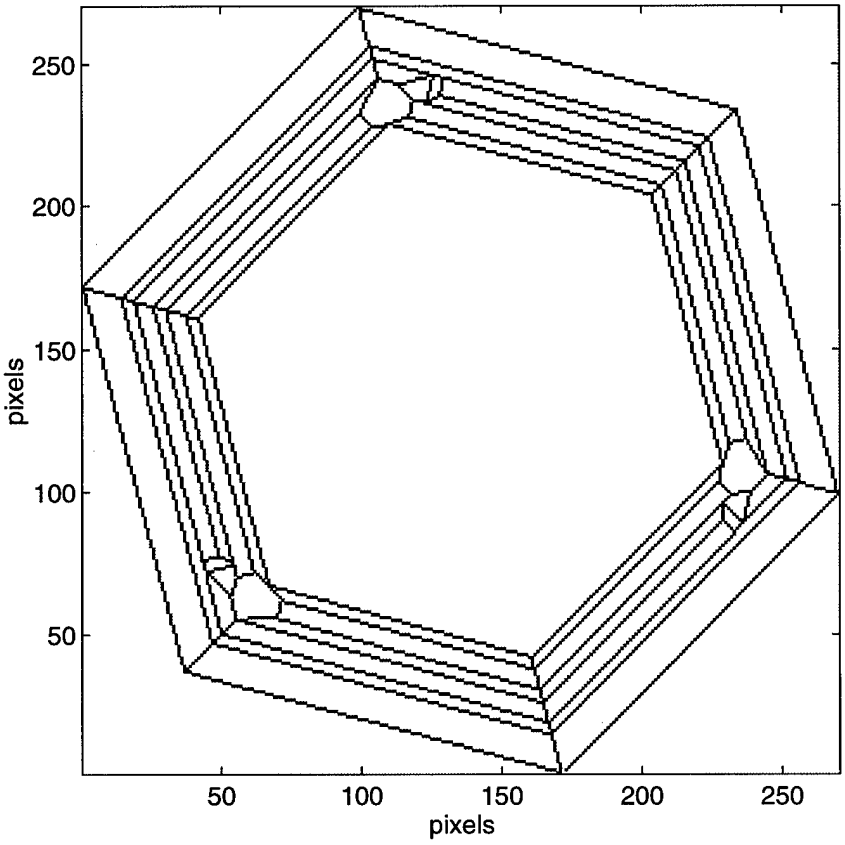

Figure 5. MUMPS9 individual micromirror line drawing.

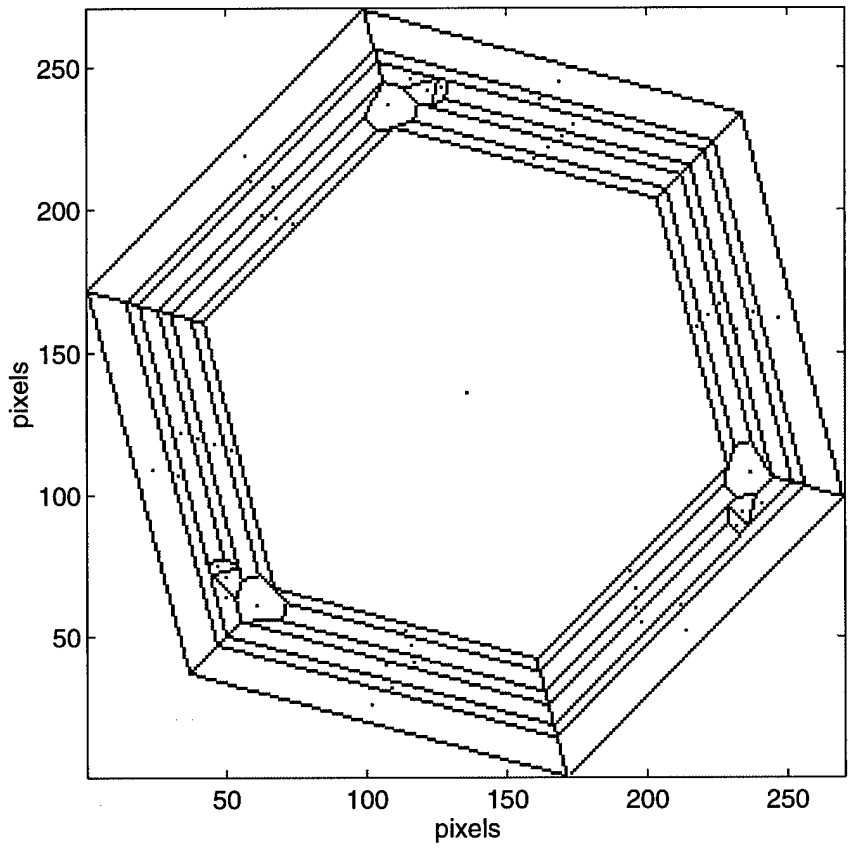

Figure 6. MUMPS9 individual micromirror line drawing with seed points.

coordinates were defined as the average of the coordinates of the polygon's vertices. Figure 6 shows the lines and seed points used to define the image of a micromirror.

2.2.1.4 Converting to pixel coordinates. All calculations to this point were performed in continuous coordinates, in units of μm. Actual drawing of the image required conversion to pixel coordinates. The conversion was accomplished by defining the extent of the image in both continuous and pixel coordinates. The physical extent of the micromirror, in both η and ξ, was found to be from -66.365μm to 66.365μm. The extent of the image in pixel coordinates was chosen to be 90×90 pixels, because this was the largest size image for an individual mirror that could be tiled into a 127-mirror hexagonal array and still fit within an overall 1024×1024

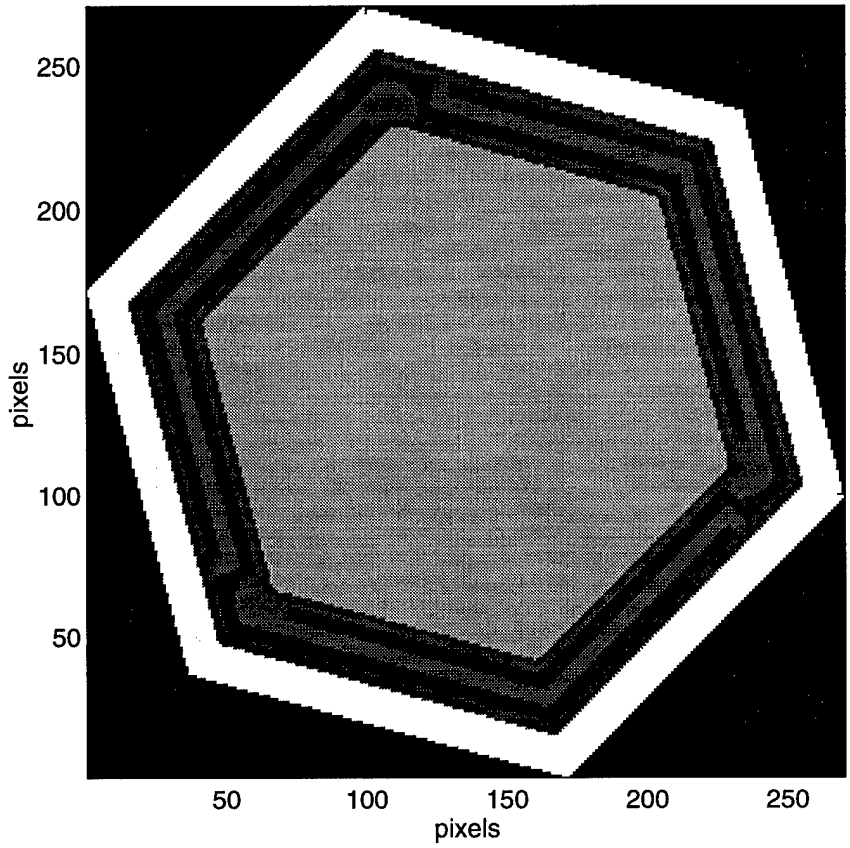

Figure 7. MUMPS9 indexed micromirror image. Brighter pixels represent a higher area index number.

image. This resulted in a sample spacing of $\Delta\xi = \frac{66.365-(-66.365)}{90-1} = 1.491\mu$m. This is less than 2μm, satisfying the Nyquist criterion of having at least two samples across the smallest structure (the flexure arms) (3).

2.2.1.5 Drawing the image. Using low-level drawing routines (`pline.m` and `pfilll.m`, see Appendix C), all the lines were drawn and all the polygons were "filled" with the index of the area that they were a part of, as defined in table 3. The result can be seen in Figure 7.

2.2.2 Tiling micromirrors into an array. Combining multiple copies of the individual micromirror image into an image of an array of micromirrors required two

steps: computing the locations of each of the mirrors, and copying the pixels of the micromirror image into the appropriate pixels of the array image.

2.2.2.1 The array mirror locations. Symmetry greatly simplified the task of locating all 127 micromirrors. Since they are set in a regular hexagonal array, going from the center of one mirror to that of one of its neighbors can be described by one of only three vectors. These vectors all have the same length, equal to the intermirror spacing of 115μm, and their directions are also easily determined from the fact that they must be $2\pi/3$ apart, and the whole array is rotated $\pi/12$. Since the array consists of six hexagonal rings around a central mirror, it is convenient to calculate the locations one ring at a time. Conveniently, the number of mirrors in each side of a hexagonal ring is equal to that ring's number, counting out from the central mirror. In other words, if the central mirror is defined to be the first "ring" (with 1 mirror to a side), then the nth ring will have n mirrors to a side. Starting by defining the location of the central mirror to be $(0, 0)$, the locations of all 127 mirrors were determined by adding the appropriate vector to the last calculated location.

2.2.2.2 Converting to pixel coordinates. Since the spatial sampling $\Delta\xi$ of the reflectivity function was already set by the choice of the size of the individual micromirror image, the conversion to pixel coordinates could not be defined here by the edges of the image. Instead, the center pixel (513,513) was defined to represent the physical location $(0, 0)$, and each pixel was defined to represent a sample point separated by $\Delta\xi$ from its neighbors.

2.2.2.3 Making the array. To make the image of the micromirror array, a blank (all pixels = 0) 1024×1024 pixel image was created. The image of the individual mirror was added to this array image at each of the mirror locations calculated previously, offset by 45 pixels to the left and 45 down, so that the centers of the mirrors fell on the calculated locations.

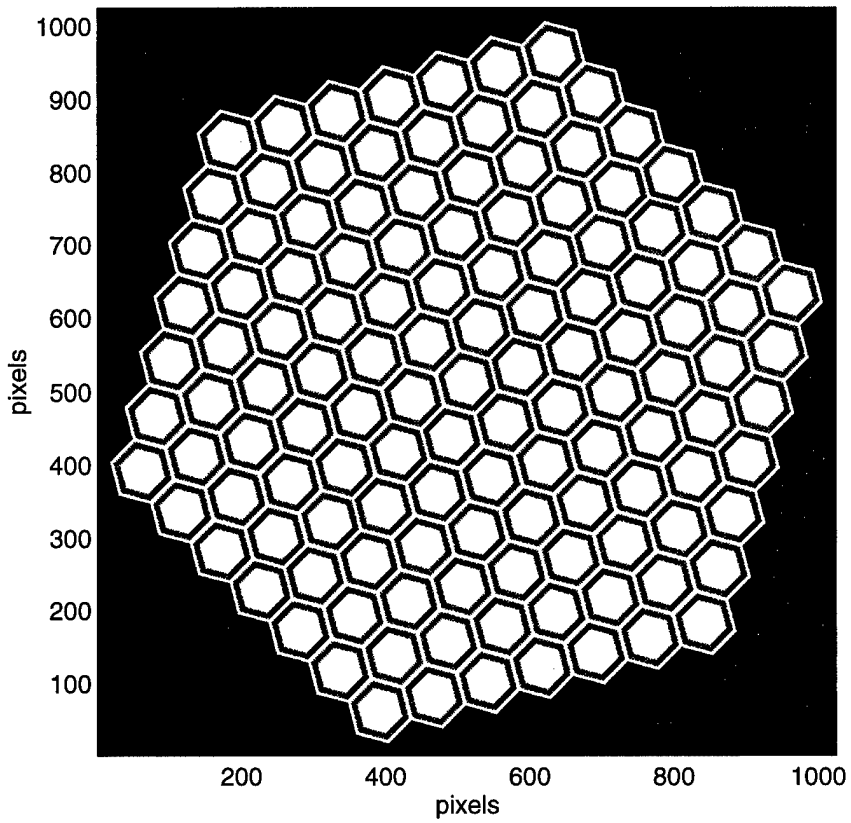

Figure 8. MUMPS9 micromirror array reflectivity image.

2.2.3 $h_{m,n}$ *and* $r_{m,n}$. The array image described above has pixel values representing the areas defined in table 3. To get the array images representing the height and reflectivity of the sample points, each pixel value must be replaced by the appropriate height or reflectivity value as determined from table 3. The purpose of using this indexing method is to avoid duplication of effort. Creating the image of the individual micromirror and tiling it into an image of an array of micromirrors are time-consuming tasks that only need to be performed once with this method. The resulting sampled absolute reflectivity and height functions are shown in Figures 8 and 9.

2.2.4 Mirror curvature. Interferometric measurements show that the individual mirrors in the array are not flat. They are curved, with a total difference in

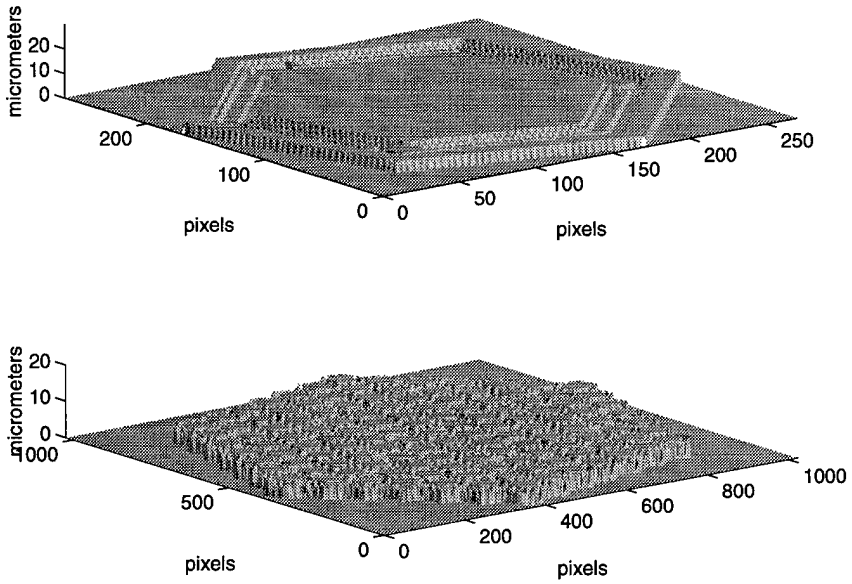

Figure 9. MUMPS9 micromirror array height image.

height between the corners and the center of the mirror of about 440nm. Since this is a non-negligible fraction of a wavelength (632.8nm), it was incorporated into the model as a modification of the height image $h_{m,n}$. The curvature was modeled as a purely parabolic profile, given by:

$$\delta h_{m,n} = 0.44(\rho_{m,n}/\rho_0)^2$$

where $\rho_{m,n} = \sqrt{\xi_m^2 + \eta_n^2} = \Delta\xi\sqrt{m^2 + n^2}$ is the distance of a point on the mirror from the center of the mirror, ρ_0 is the distance from the center of the mirror to the corners (50μm), and δh is the amount which must be subtracted from $h_{m,n}$ to impart the parabolic curving profile. To ensure that the correction δh is applied only to the mirror surface, and not the flexures or anchors, it must be multiplied by a hexagonal mask function that is 1 for points on the mirror and 0 elsewhere.

Applying the correction to the height function used a process similar to that used to create the height function in the first place. First, a 90 $\times$ 90 pixel sampled

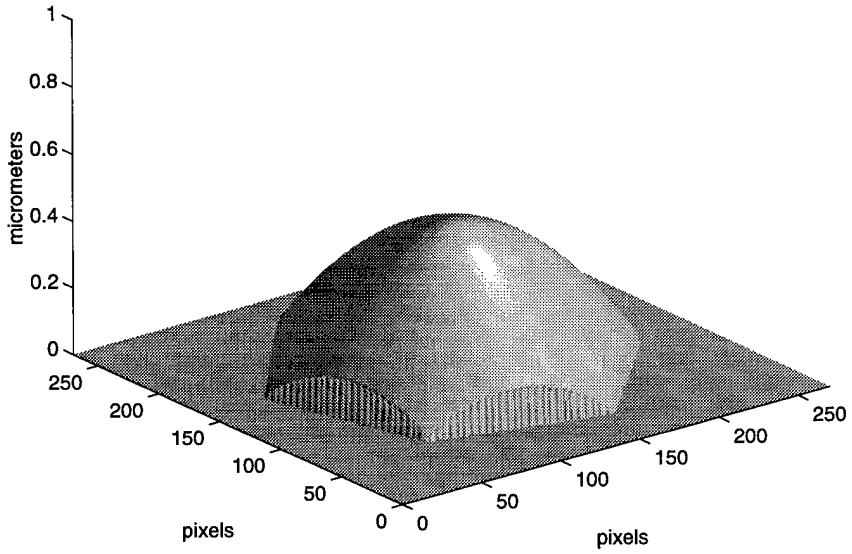

Figure 10. MUMPS9 micromirror curvature correction.

version of $\delta h_{m,n}$ is created. The micromirror movable surface image described in the next section provided a hexagonal mask. The result is shown in Figure 10, where the height of a point represents the amount to be *subtracted* from the corresponding point of the height image. This image was then tiled in the same way as the mirror index image, as described previously. Subtracting these curvature correction images from the single mirror and mirror array height images produced the modified height images of Figure 11.

2.2.5 Deflecting individual mirrors. Simulating the deflection of individual micromirrors is accomplished by making the appropriate adjustments to the height image $h_{m,n}$. Usually, the amount of deflection for each mirror depends on that mirror's location, so a look-up table of these locations will be useful. Each mirror in the array image consists of many pixels, and each of these pixels must be adjusted the same way for a given mirror. It will therefore be necessary to have a way of

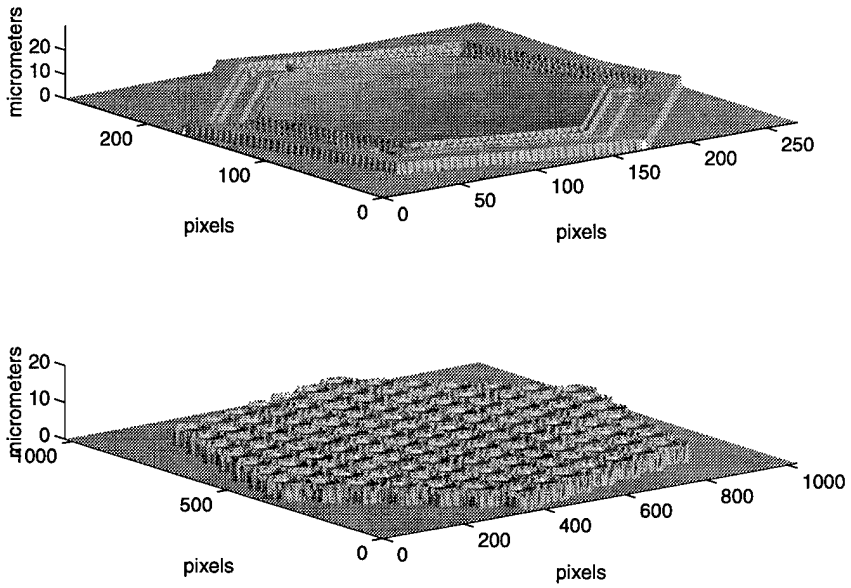

Figure 11. MUMPS9 height images, with curved mirrors.

determining not only whether a given pixel moves, but also which mirror it belongs to. This is the purpose of the look-up image.

2.2.5.1 The look-up image. The look-up image is the same size as the array height image, but the pixel values have different meanings. A pixel value of 0 indicates that the corresponding pixel in the height image represents a sample point that does not move. A pixel value of $k \neq 0$ indicates that the corresponding height image pixel represents a point that does move, and that the point belongs to mirror k. This look-up image is constructed in a manner similar to the indexed array image.

First, a 90×90 pixel Boolean (1: moves, 0: does not move) image of an individual movable surface is constructed. This is similar to the image of an individual micromirror, but much simpler. There are only six vertices in this image, and six lines. This image, as shown in Figure 12, represents the points on the physical de-

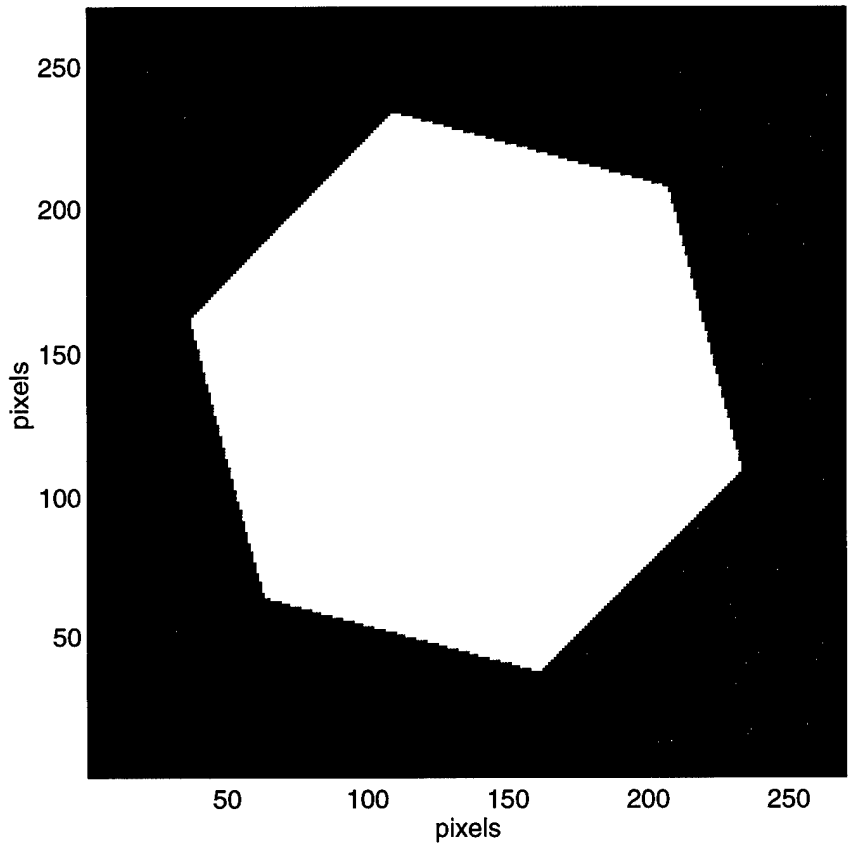

Figure 12. MUMPS9 micromirror movable surface image.

vice that are either on the gold-plated mirror surface or the 2μm wide Poly2 edge surrounding it.

This image is then tiled into an array image, with a slight difference when compared to the way the indexed image was constructed. Before being added to the array image, each individual movable surface image is multiplied by the index in the look-up table that corresponds to the mirror being added.

2.2.5.2 Modifying the height image. Since the pixels in the look-up image correspond directly to those in the height image, simulating control inputs is relatively simple:

1. Scan each pixel in the look-up image.

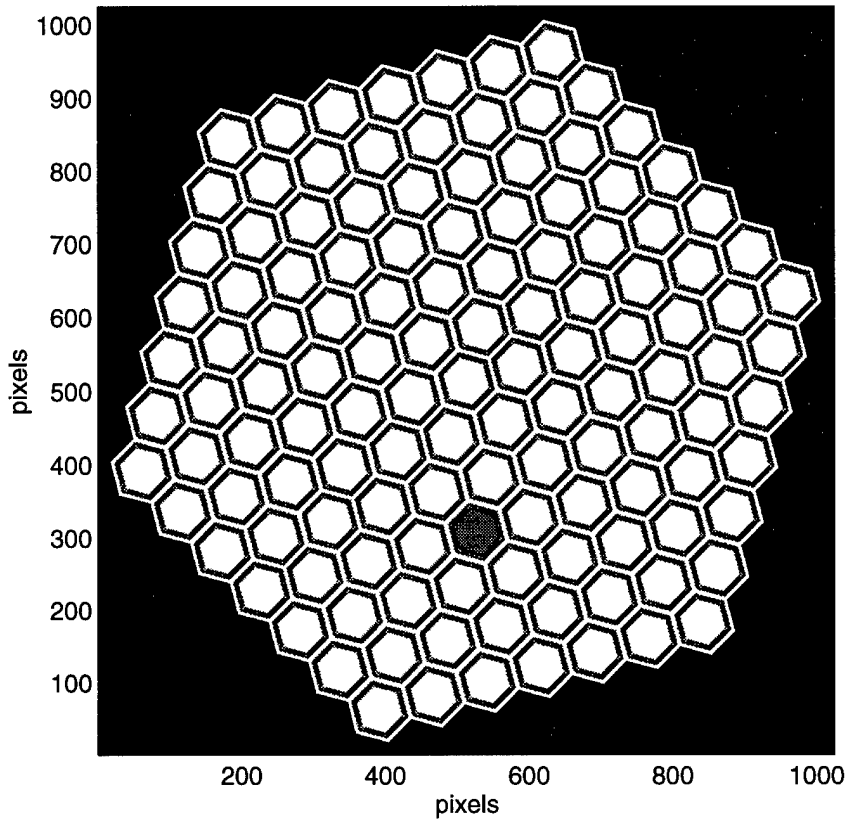

Figure 13. MUMPS9 micromirror array height image, single mirror deflected.

2. For each pixel that has a nonzero value k, look up the location of the center of the kth mirror in the look-up table.
3. Calculate the amount of deflection required based on mirror location.
4. Subtract the appropriate amount from the corresponding pixel in the height image.

This method was used to deflect a single mirror (Figure 13), to add an overall tilt to the mirror array (Figure 14), and to "cup" the array with a parabolic profile (Figure 15).

2.2.6 $R_{m,n}$. Once the height image $h_{m,n}$ is adjusted to simulate any desired deflection of the micromirrors, it can be combined with the absolute reflectivity

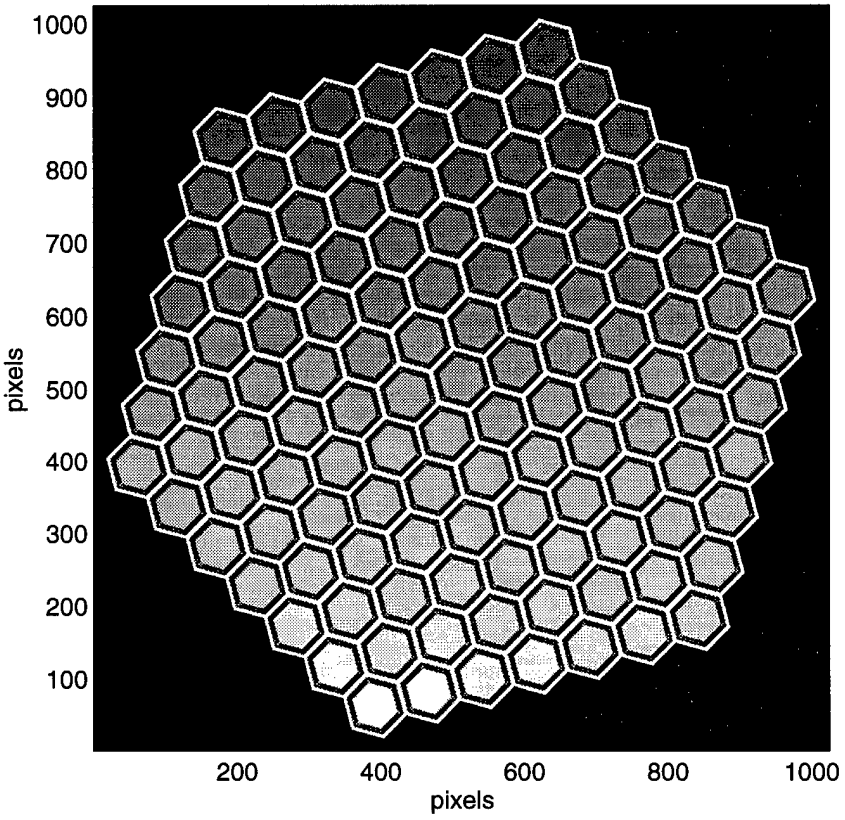

Figure 14. MUMPS9 micromirror array height image, tilted profile.

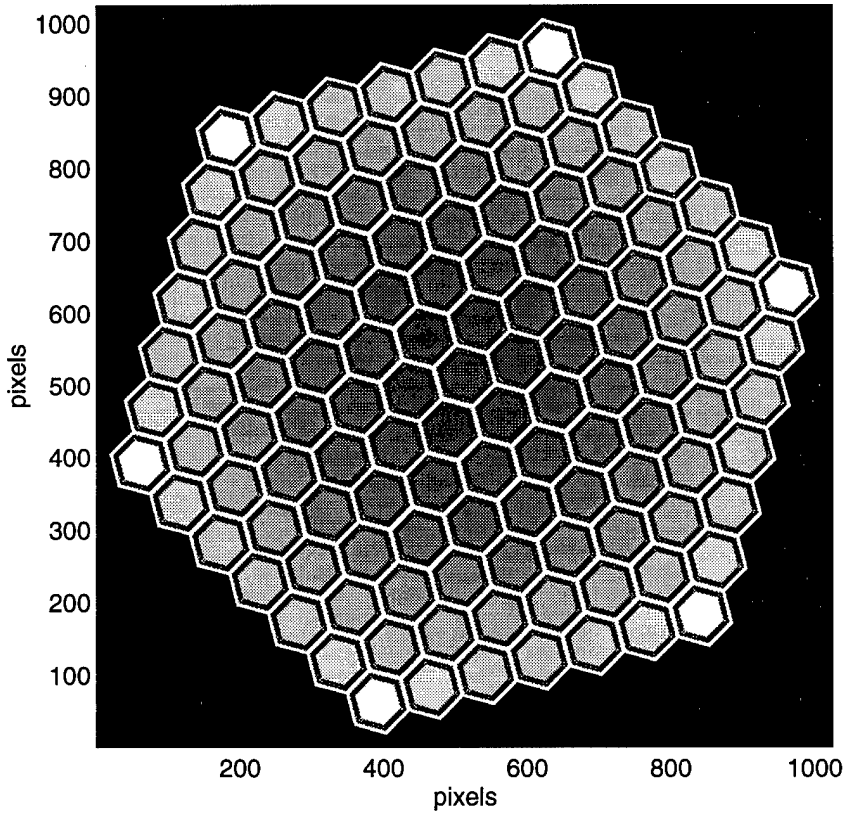

Figure 15. MUMPS9 micromirror array height image, parabolic profile.

image $r_{m,n}$ to form the complex reflectivity image $R_{m,n}$. Because the MEMS device operates by reflection, an incoming wavefront undergoes a round trip through the varying height across the device before leaving. Also, a positive height will cause the phase at that point to be advanced (positive phase) in relation to a plane at $h = 0$. The phase of $R_{m,n}$ will therefore be equal to $2\pi(2h_{m,n}/\lambda)$, leading to the following definition:

$$R_{m,n} = r_{m,n}exp(i4\pi h_{m,n}/\lambda).$$

2.3 The wavefront function $W_{m,n}$

The next element of the model is the wavefront function $W_{m,n}$. This is a complex-valued function which can be considered an image of the wavefront. Only wavefronts of constant, unit intensity were modeled. Aberrations were modeled as a variation of the wavefront's phase across the image.

2.3.1 Plane wave. A plane wave is very easy to model, since it has constant phase across a plane:

$$W_{m,n} = 1.$$

2.3.2 Spherical wave. What is spherical is the location of a surface of constant phase, so it is easiest to define $z_{m,n}$, the height of a spherical surface above a reference plane, then:

$$W_{m,n} = exp(i2\pi z_{m,n}/\lambda).$$

Obviously, z will depend only on the distance ρ of a pixel from the center of the image.

For a point (ρ, z) on the sphere, $R^2 = \rho^2 + (z - d + R)^2$, where R is the radius of curvature of the wavefront and d is the total phase difference at the center of the image. Solving for z and keeping only the physical solution, the above equation can be rewritten $z = d - R + \sqrt{R^2 - \rho^2}$. The two variables which can be used to

characterize the spherical surface are d and R, and it will be useful to be able to use either one. Fortunately, they are related, and each can be calculated from the other, assuming that zero phase change occurs at the edge of the image. Setting $z = 0$ at $\rho = \rho_{max}$ provides the following relationships:

$$d = R - \sqrt{R^2 - \rho_{max}^2},$$

$$R = \frac{d^2 + \rho_{max}^2}{2d}.$$

Given d and $\Delta\xi$, then, the first step in creating an image of a spherical wavefront is to make a 1024×1024 pixel image. The pixel $(513, 513)$ is defined as the center pixel ($\rho = 0$). Since z is a function of ρ^2, it is convenient to set each pixel to the value of ρ^2 for that pixel, given by

$$\rho_{m,n}^2 = \left((m - 513)^2 + (n - 513)^2\right) \Delta\xi^2.$$

ρ_{max} is then simply $\rho_{1,1}$. This allows calculation of either

$$R = \frac{d^2 + \rho_{1,1}^2}{2d}$$

or

$$d = R - \sqrt{R^2 - \rho_{1,1}^2}$$

depending on whether the radius of curvature or the total central phase difference is specified. These are then used to calculate

$$z_{m,n} = d - R + \sqrt{R^2 - \rho_{m,n}^2}.$$

2.4 *The source optical disturbance $U_{m,n}$*

The optical disturbance in the source plane is the result of the incoming wavefront being reflected off the MEMS device. Mathematically, it is defined as

$$U(\xi, \eta) = R(\xi, \eta) \times W(\xi, \eta).$$

This is the function whose Fourier transform will be calculated, in order to predict the far-field diffraction pattern.

2.5 *The diffraction intensity $I_{u,v}$*

Fourier optics provides a convenient way to calculate the distribution of optical energy in an observation plane from the distribution in a source plane. For the sake of this discussion, the source plane will be defined by $z = 0$ in Cartesian coordinates, while the observation plane will be defined by some fixed $z \neq 0$. The coordinates of a point in the source plane will be written (ξ, η), and those of a point in the observation plane will be (x, y), as shown in Figure 16. Only those results of Fourier optics that are required for the model are covered here. For a more detailed explanation, see Goodman (11).

2.5.1 Scalar Diffraction Theory. Scalar diffraction theory has been found to accurately represent the effects of optical wavefront propagation when certain easily realized conditions are met. If the order of the size of the diffracting object is large compared to the wavelength λ, and the observation of the diffraction pattern is carried out far from the object, then light can be treated as a complex scalar instead of as a vector field. Since complex numbers have both amplitude and phase, the wave-like nature of light is maintained by the new representation, and the math becomes simpler. The "optical disturbance" $U_1(x, y)$ at the point (x, y, z) in the observation plane defined by z is simply the sum of the contributions from each point $(\xi, \eta, 0)$ in

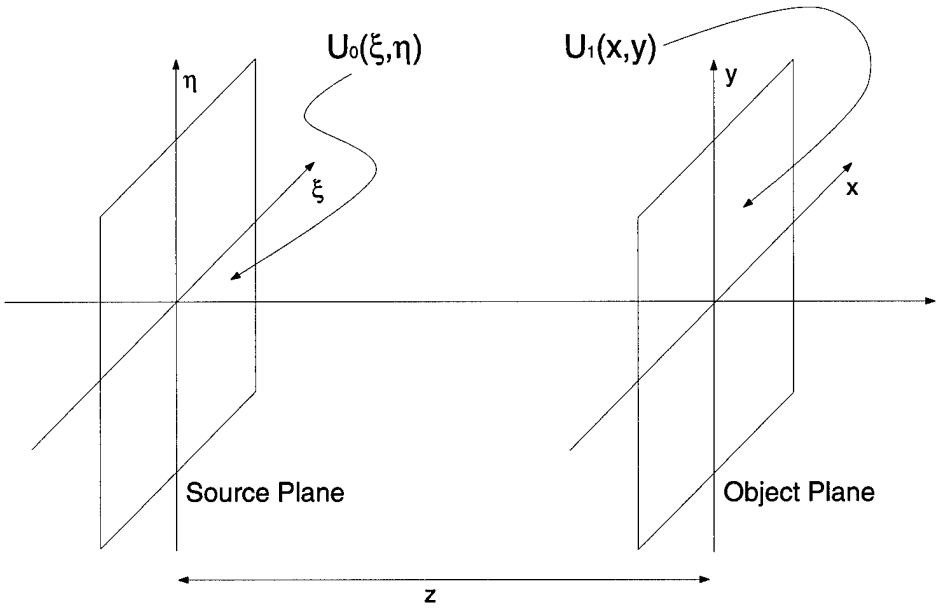

Figure 16. Notation used to describe scalar diffraction.

the source plane at $z = 0$, according to the Huygens-Fresnel principle (11):

$$U_1(x,y) = \frac{z}{i\lambda}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} U_0(\xi,\eta)\frac{exp(ikr_{01})}{r_{01}^2}d\xi d\eta \tag{1}$$

where $k = 2\pi/\lambda$ and r_{01} represents the distance from the source point $(\xi, \eta, 0)$ to the observation point (x, y, z). In other words,

$$r_{01}^2 = z^2 + (x-\xi)^2 + (y-\eta)^2. \tag{2}$$

2.5.2 The Fresnel and Fraunhofer approximations. Equation 2 can be rewritten as

$$r_{01} = z\sqrt{1 + \left(\frac{x-\xi}{z}\right)^2 + \left(\frac{y-\eta}{z}\right)^2}. \tag{3}$$

Taking a binomial expansion of equation 3 and keeping only quadratic and lower-order terms constitutes the Fresnel approximation:

$$r_{01} \simeq z\left[1+\frac{1}{2}\left(\frac{x-\xi}{z}\right)^2+\frac{1}{2}\left(\frac{y-\eta}{z}\right)^2\right]. \tag{4}$$

This approximation will lead to results valid near the optical axis. Inserting equation 4 into equation 1 and rearranging terms leads to the Fresnel formulation of diffraction:

$$U_1(x,y)=\frac{exp(ikz)}{i\lambda z}exp\left(i\frac{k}{2z}\left(x^2+y^2\right)\right)\times\mathcal{F}\left\{U_0(\xi,\eta)exp\left(i\frac{k}{2z}\left(\xi^2+\eta^2\right)\right)\right\}\Big|_{\omega_\xi=x/\lambda z,\omega_\eta=y/\lambda z}, \tag{5}$$

where $\mathcal{F}\{\bullet\}$ is the Fourier transform (FT) operator. The optical disturbance in the observation plane is proportional to the FT of the optical disturbance in the source plane multiplied by a quadratic phase factor (11).

Even though Fresnel diffraction is also called "near field" diffraction, equation 5 is also valid in the far field. However, the quadratic phase factor complicates calculations, so a further simplification is usually made for the far field. If $z \gg k(\xi^2+\eta^2)_{max}/2$, (the observation plane is far from the source plane), then the quadratic phase term inside the Fourier transform operator can be approximated by unity and removed from equation 5. This produces the Fraunhofer, or "far field" diffraction equation (11):

$$U_1(x,y)=\frac{exp(ikz)}{i\lambda z}exp\left(i\frac{k}{2z}\left(x^2+y^2\right)\right)\times\mathcal{F}\left\{U_0(\xi,\eta)\right\}\Big|_{\omega_\xi=x/\lambda z,\omega_\eta=y/\lambda z}. \tag{6}$$

The intensity of the light in the observation plane is given by $I(x,y)=|U(x,y)|^2$, so the complex values outside the Fourier transform in equation 6 reduce to $1/\lambda^2 z^2$.

2.5.3 The Fourier transform. The Fourier transform, $F(\omega)$, of a function $f(x)$ is defined by

$$F(\omega) = \int_{-\infty}^{\infty} f(x) exp(-i2\pi x\omega) dx. \tag{7}$$

In the case of a two dimensional function $f(x, y)$, the definition is

$$F(\omega_x, \omega_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) exp(-i2\pi (x\omega_x + y\omega_y)) dx dy. \tag{8}$$

The FT of a function is also called the spectrum of the function. In fact, one of the more useful properties of the Fourier transform is to translate a function from the time domain to the frequency domain. If x is a spatial dimension and $f(x)$ represents the spatial distribution of some property (like reflectivity), then ω_x represents the spatial frequency associated with x, and $F(\omega_x)$ relates the distribution of the same property described by $f(x)$, but distributed in the frequency domain.

2.5.4 The Discrete Fourier transform. In the case of a computer model, it is usually not possible to define $f(x)$ as an analytic function of x. Instead, $f(x)$ is sampled at discrete intervals Δx a finite number of times, N. The discrete Fourier transform (DFT) of a sampled function is defined by Gonzalez (10):

$$F_u = \frac{1}{N} \sum_{n=0}^{N-1} f_n exp(-i2\pi un/N) \tag{9}$$

where if the samples f_n are separated by the interval Δx on the x-axis, then the F_u's correspond to frequency domain samples on the ω-axis separated by

$$\Delta\omega = \frac{1}{N\Delta x}. \tag{10}$$

For the two-dimensional case, the appropriate equation is:

$$F_{u,v} = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f_{m,n} exp\left(-i2\pi \left(\frac{um}{M} + \frac{vn}{N}\right)\right). \tag{11}$$

One can see from equation 9 that for a one-dimensional array of N samples f_n, N complex multiplications and N complex additions are required to calculate each of the N samples F_u (10:119). This means that a total of N^3 calculations are required to compute an entire one-dimensional DFT, using equation 9. However, if N is an integer power of 2, there exist algorithms called fast Fourier transforms (FFTs) that take advantage of symmetry to make do with only $N\log_2 N$ complex calculations (10:119) per sample F_u, for a total of $N^2\log_2 N$ complex calculations for the entire transform. A two-dimensional DFT is typically implemented by computing the one-dimensional DFT of each row, followed by the one-dimensional DFT of each column of the result (10:94). If each of these two steps requires N^3 calculations for each of the N rows or columns, then the total number of calculations required will be

$$2 \times N \times N^3 = 2N^4,$$

using equation 11. If, on the other hand, FFT algorithms are used, the number of required calculations drops to

$$2 \times N \times N^2\log_2 N = 2N^3\log_2 N.$$

2.5.5 The classical intensity $I(x, y)$. The diffraction intensity pattern in the observation plane, $I(x, y)$, is given by

$$I_1(x, y) = |U_1(x, y)|^2.$$

This is implemented by computing the FFT of $U_{n,m}$, squaring the absolute value of the result, and normalizing it.

2.6 *Spatial sampling of the diffraction pattern*

The source disturbance function $U_{m,n}$ is a sampled function, with sample points separated by $\Delta\xi$ measured in μm. Its Fourier transform, as shown by equation 10, will have sample points separated by $\Delta\omega$ measured in μm^{-1}. Equation 6 shows how to calculate the spatial sampling Δx of the diffraction pattern:

$$\Delta x = \lambda z \Delta\omega = \frac{\lambda z}{N\Delta\xi}.$$

Since z is not necessarily known, the diffraction pattern can also be treated as an angular spectrum, with angular sampling $\Delta\alpha = \lambda/N\Delta\xi$ measured in radians. In this case, the spatial sampling can be determined when a value is chosen for z by calculating $\Delta x = z\Delta\alpha$.

2.7 *Summary*

The MUMPS9 model was constructed in parts. First, both the height and reflectivity across an image of an individual micromirror were sampled. This sampled function was then tiled into an array of 127 identical micromirrors. Methods were developed to deflect the mirrors individually, to either introduce or correct an aberration in a reflected wavefront. Sampled representations of an incoming and a reflected wavefront were also formed. The application of an FFT algorithm to the sampled reflected wavefront ultimately produced a sampled, predicted diffraction pattern.

III. Results

This section discusses the results of a series of simulation runs. Because micromirror arrays are proposed for use as a deformable mirror in an adaptive optics system, the main area of interest is to simulate the correction of an aberrated wavefront using a MUMPS9 device. First, to determine whether controlling the phase of a wavefront using a MUMPS9 device is feasible, correction of a spherically aberrated wavefront was simulated. The effect of the curvature of the individual micromirrors on the effectiveness of the device was studied. Next, for validation purposes, duplication of measurements made in the laboratory with a physical MUMPS9 device was attempted. Finally, the use of a lenslet array to reduce diffraction effects is also simulated.

3.1 Simulation of Correction of an Aberrated Wavefront

To measure the effectiveness of using a MUMPS9 device to correct a fixed aberration, three runs are required. First, the pattern resulting from reflection of a plane wave by an undeflected MUMPS9 device must be simulated; this results in the theoretical ideal, the yardstick with which other runs will be measured. Next, the result of reflection of an aberrated wave by an undeflected MEMS must be calculated, to show the effect of introducing the aberration. Finally, reflection of an aberrated wave by a MEMS deflected appropriately to correct the aberration must be simulated. When compared with the first two runs, the final one will show how effective the device is at correcting aberrated wavefronts. These three runs require two versions of the complex reflectivity function (one undeflected, the other with corrective deflection) and two wavefront models (one planar, one aberrated). Simulation of the use of a MUMPS9 device to correct a spherical aberration was performed using the complex reflectivity function described in the previous chapter. The first run involved reflecting a plane wave on an undeflected MUMPS9 mirror

array with flat mirrors. The resulting normalized log diffraction pattern is shown in figure 17. In order to provide better visualisation of the effects of the aberration and the correction, all diffraction pattern images in this chapter were formed by converting each pixel of the predicted intensity pattern to its value in dB, normalized to the peak intensity value. The result was then thresholded at -60dB, renormalized to a scale of 1 to 256, and displayed as a grayscale image with 256 gray levels. So in all diffraction pattern images, a white pixel corresponds to a point in the pattern with an intensity 60dB or more below the peak intensity, while a black pixel represents the peak intensity that defines 0dB, and intermediate gray levels vary linearly with the intensity expressed in dB. The effect of this was to enhance the off-axis details, while keeping the lower intensity features from washing out the patterns of interest. The aberrated and corrected runs were performed using a spherical aberration defined so that the total phase difference at the center of the array is 2π, and 0 at the corners (figure 18), using the wavelength of a HeNe laser (0.6328 μm). The predicted normalized log diffraction patterns for these simulations are shown in figures 19 and 20. The average radial distribution of energy in the central section of the diffraction patterns, normalized to the peak of the ideal conditions of the first run, is shown in figure 21. The peak intensity of the aberrated pattern is equal to 54.51% of that of the unaberrated pattern, which corresponds to a drop of -2.63dB in peak intensity. The corrected pattern brings this fraction back up to 79.42% (-1.00dB). When correcting a fixed spherical aberration with a MUMPS9 device, the theoretical limit is a minimum loss of 1.00dB in peak intensity. From a qualitative point of view, the diffraction pattern for the aberrated, uncorrected run appears "blurry" when compared to that of the unaberrated run, while the output of the aberrated, corrected run looks more like the unaberrated output.

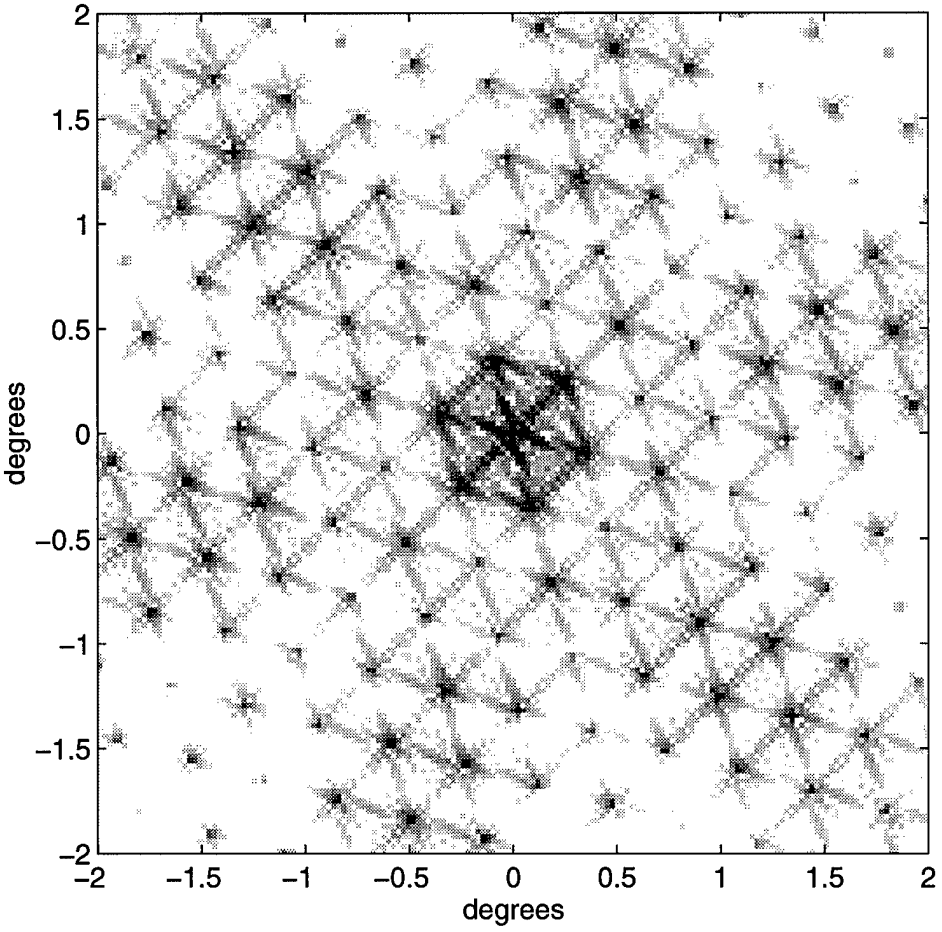

Figure 17. Diffraction pattern for a plane wave reflected from an undeflected MUMPS9 array.

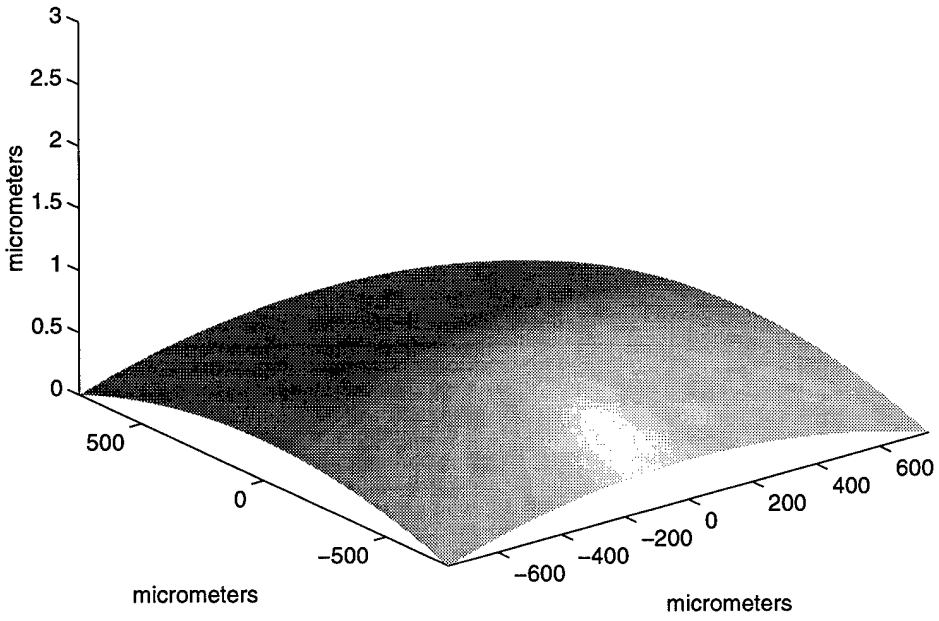

Figure 18. Spherical wavefront function.

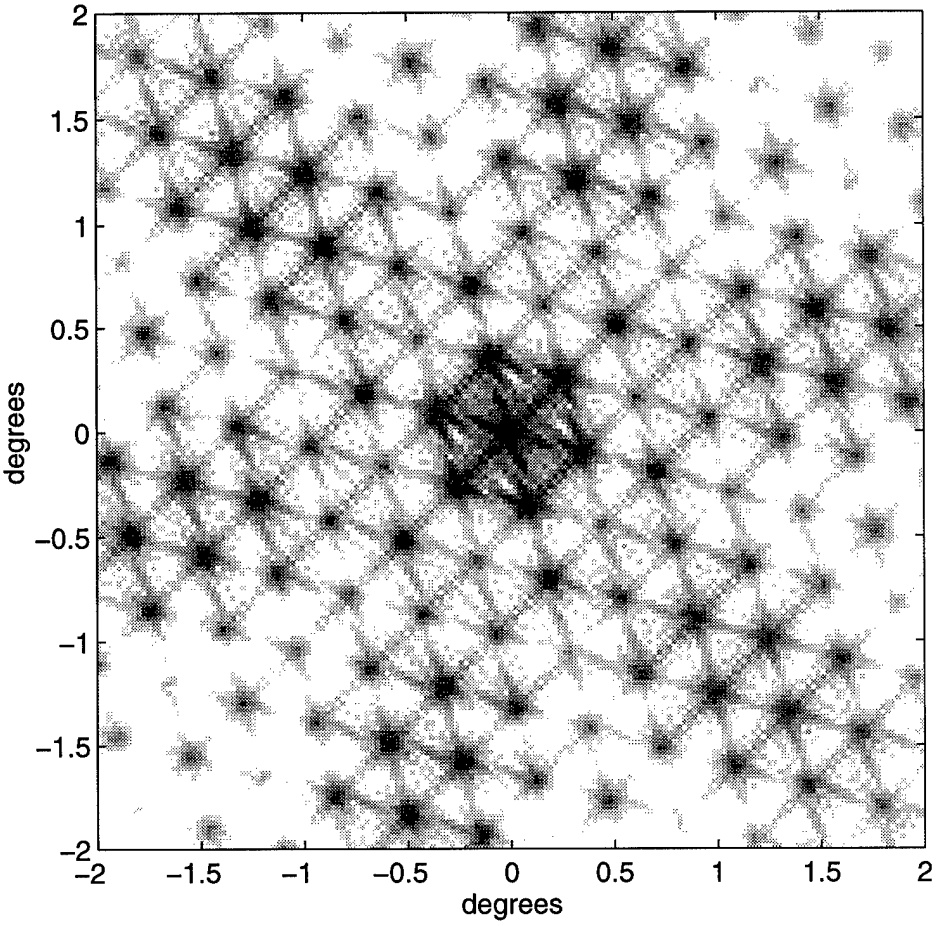

Figure 19. Diffraction pattern for a spherical wave reflected from an undeflected MUMPS9 array.

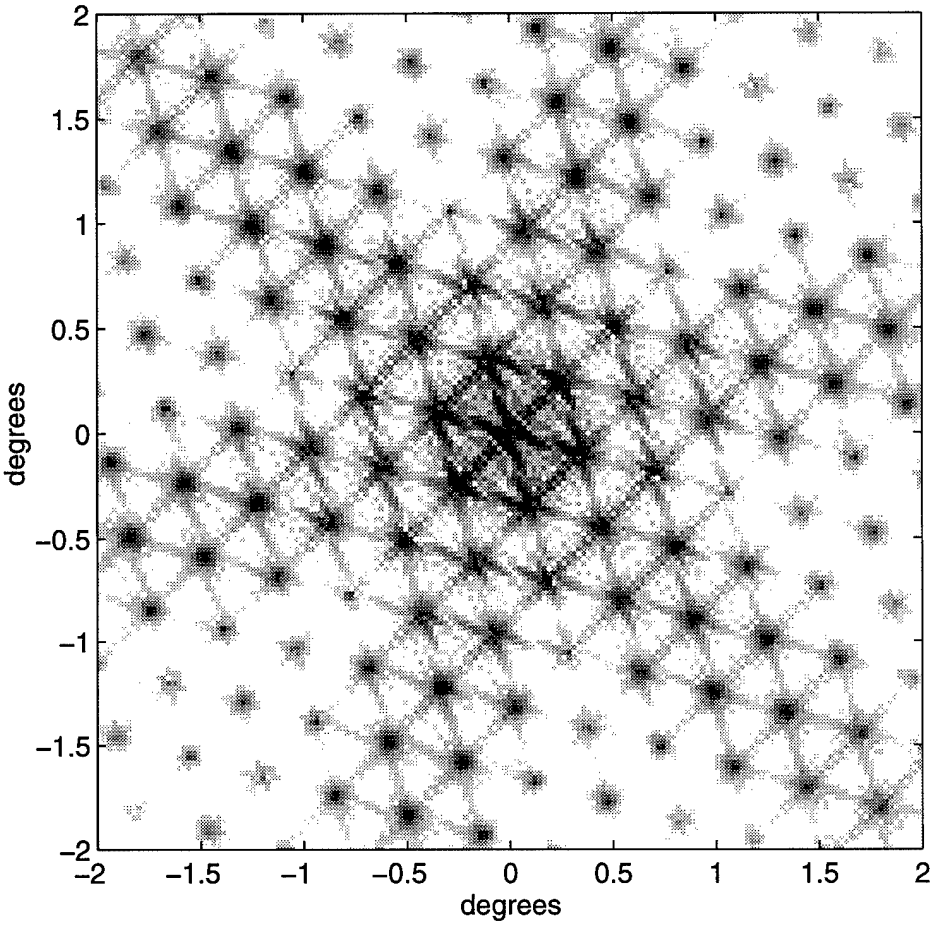

Figure 20. Diffraction pattern for a spherical wave corrected by reflection from a MUMPS9 array.

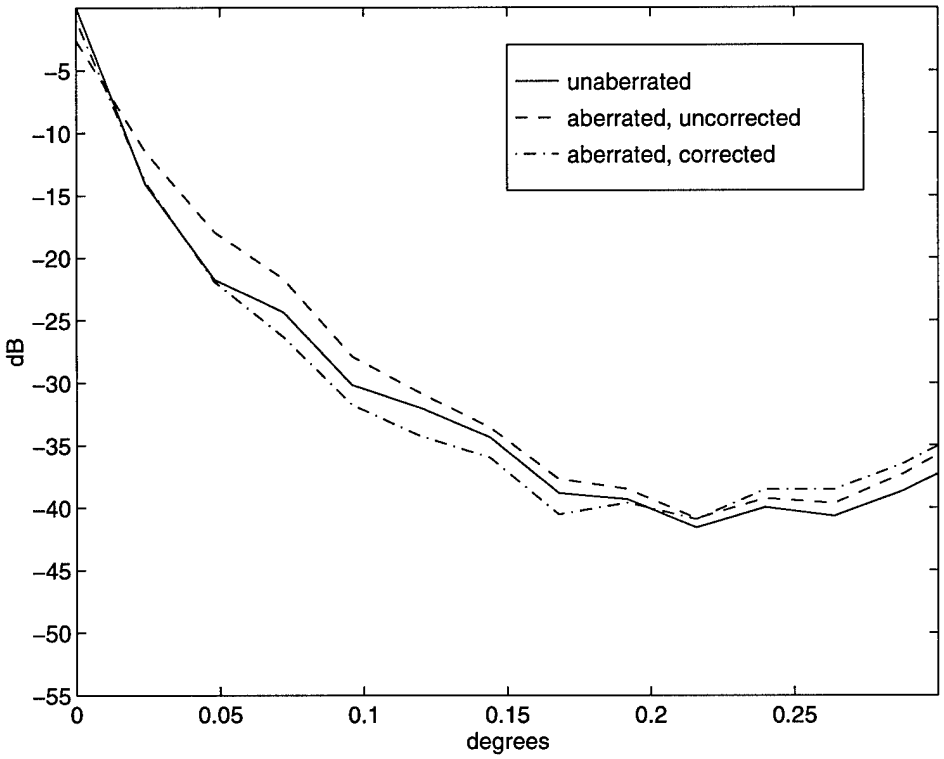

Figure 21. Radial energy distribution for correction of a spherical aberration using a MUMPS9 array, normalized to the peak intensity of the unaberrated diffraction pattern. The peak of the uncorrected aberrated pattern is at -2.63dB; that of the corrected aberrated pattern is at -1.00dB.

3.2 Effect of micromirror curvature

As shown in Figure 11, the individual micromirrors are not actually as flat as they are designed to be. The effect of this curvature is to greatly reduce the peak intensity of the diffraction pattern, although correction of an aberrated wavefront is still possible. The same three simulation runs that were described in the previous section were performed for the curved mirror model. Figures 22, 23 and 24 show the resulting depicted diffraction patterns. Figure 25 is a comparison of the average radial distribution of energy in the central section of the diffraction patterns from the two unaberrated runs. With no aberrations present, the effect of the mirror curvature is to reduce the peak intensity to 2.86% of the flat mirror value, for a drop of -15.44dB. Introducing the spherical aberration causes a further -2.99dB drop, which the curved mirrors are only able to correct back up to a -1.37dB drop relative to the curved mirror/flat wavefront peak. This is depicted in the average radial energy distribution of the curved mirror results, Figure 26, where the 0dB point is still defined as the peak intensity for the diffraction pattern from reflection of an unaberrated wavefront on an array of flat mirrors.

3.3 Comparison of Simulation to Laboratory Results

The laboratory setup used to measure the diffraction pattern from a MUMPS9 device is shown in figure 27. The output beam of a HeNe laser is expanded, resulting in a coherent wavefront that is planar on the scale of the MUMPS9 device. A mirror and a beam splitter direct this plane wavefront through a lens pair that provides a magnification of $\frac{1}{5.75}\times$ from plane B to plane A. A diverging lens can be placed in the optical path just after the beam expander to introduce a known spherical aberration. After reflection by the MUMPS9 device, the beam goes back through the lens pair, this time with a magnification of $5.75\times$. The net effect is that of reflecting the wavefront on a virtual MUMPS9 device that is 5.75 times the size of the actual device, and is located in plane B. The optical disturbance at plane B is

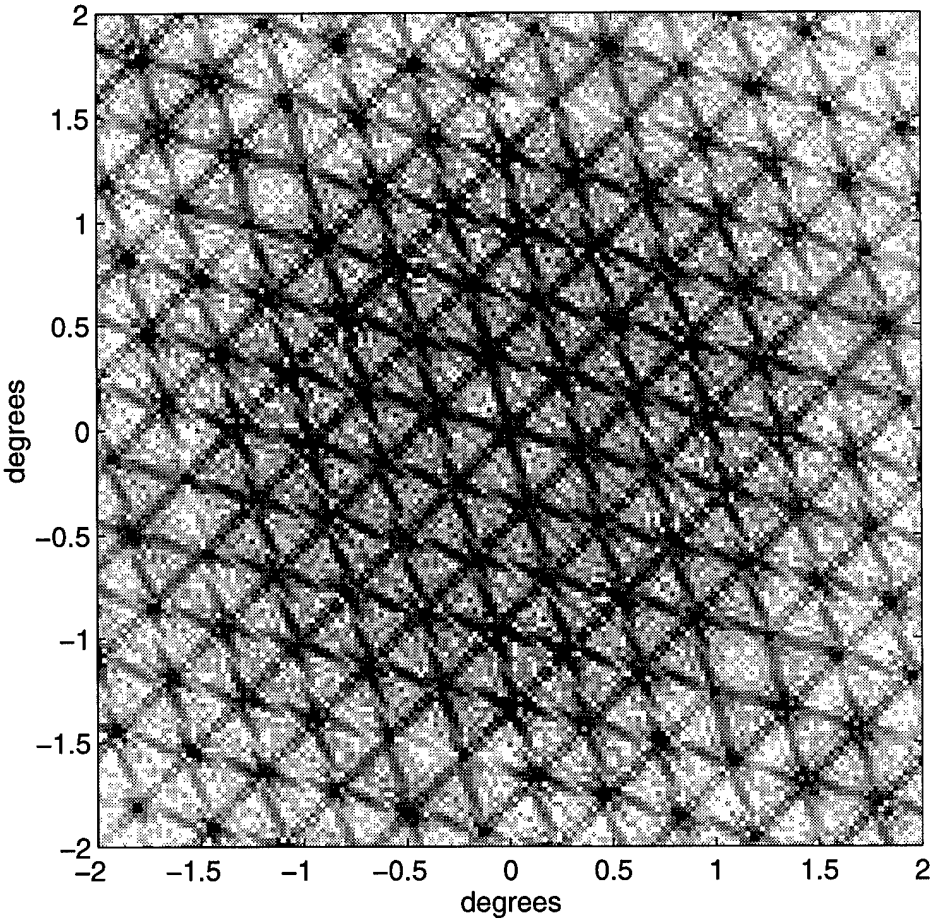

Figure 22. Diffraction pattern for a plane wave reflected from an undeflected MUMPS9 array with curved mirrors.

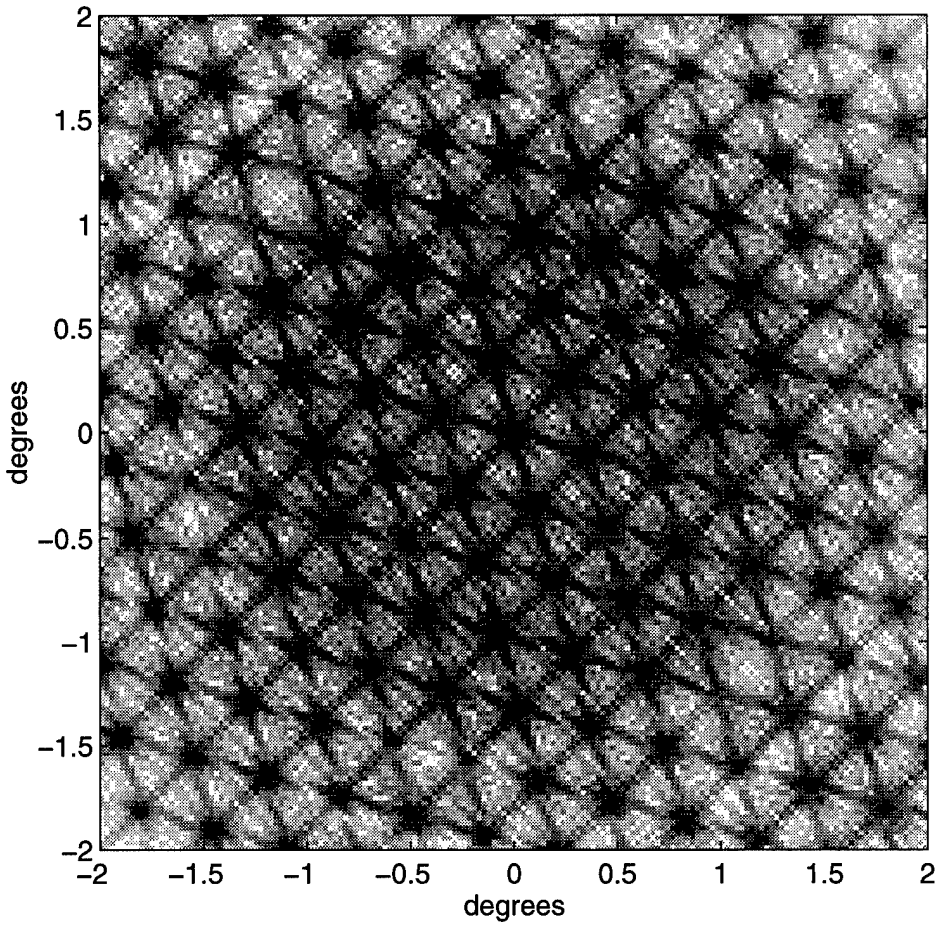

Figure 23. Diffraction pattern for a spherical wave reflected from an undeflected MUMPS9 array with curved mirrors.

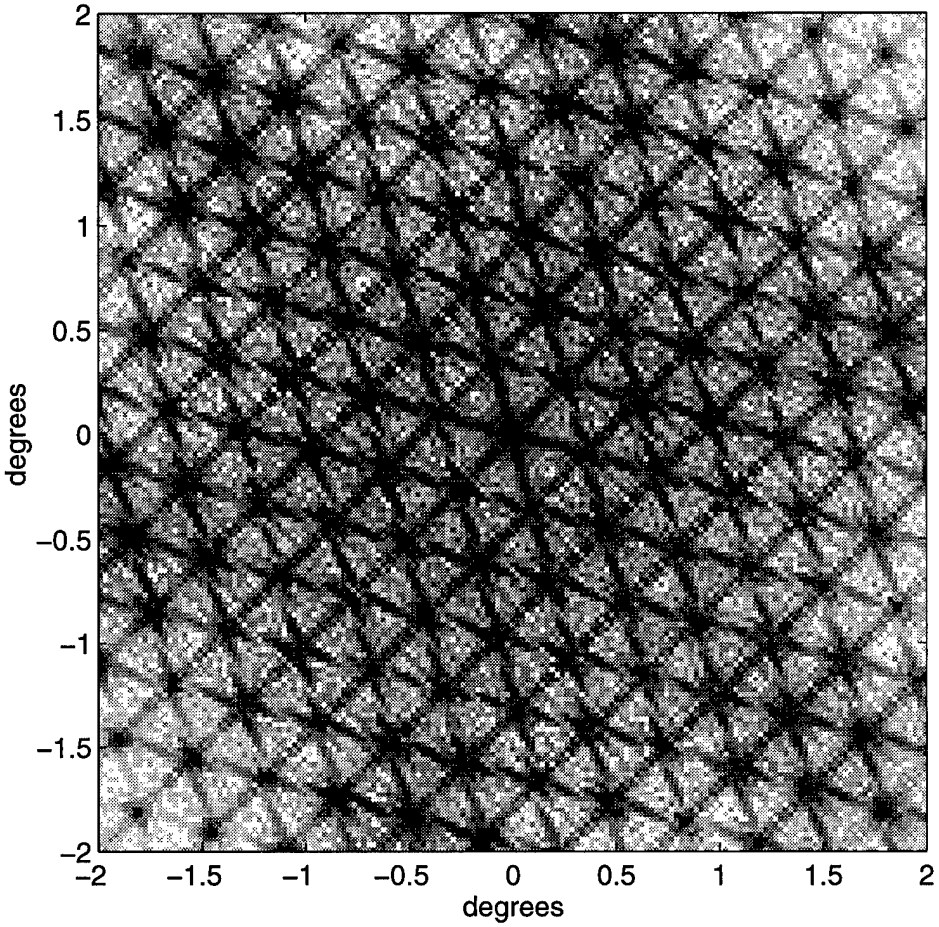

Figure 24. Diffraction pattern for a spherical wave corrected by reflection from a MUMPS9 array with curved mirrors.

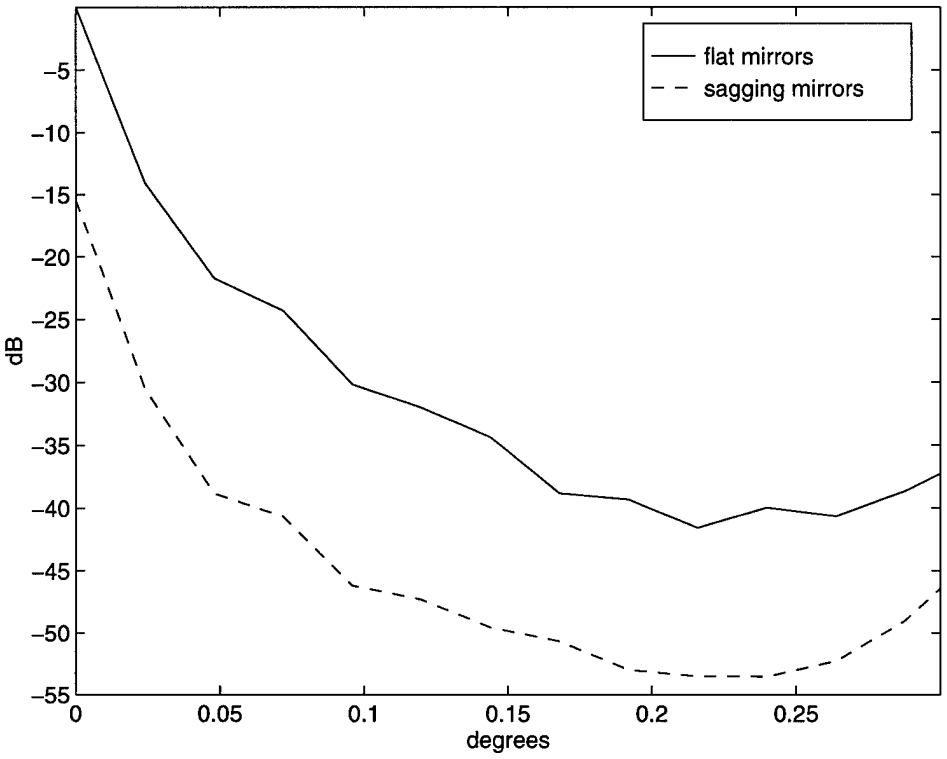

Figure 25. Comparison of the radial energy distribution for reflection of an unaberrated wavefront on a MUMPS9 device with flat and curved mirrors, normalized to the peak intensity of the flat mirror case. The peak intensity of the curved mirror case is at -15.44dB.

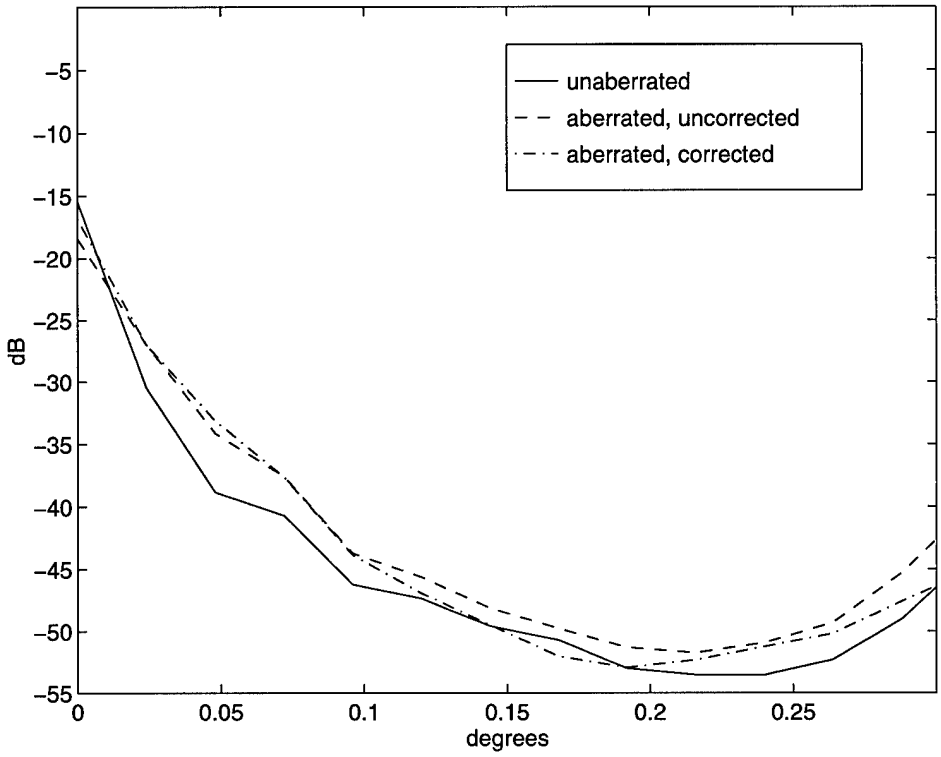

Figure 26. Radial energy distribution for correction of a spherical aberration using a MUMPS9 array with curved mirrors, normalized to the unaberrated, flat mirror case. As shown above, the peak intensity of the unaberrated, curved mirror case is at -15.44dB. The peak of the uncorrected aberrated curved mirror case is at -18.43dB, and that of the corrected aberrated curved mirror case is at -16.81dB.

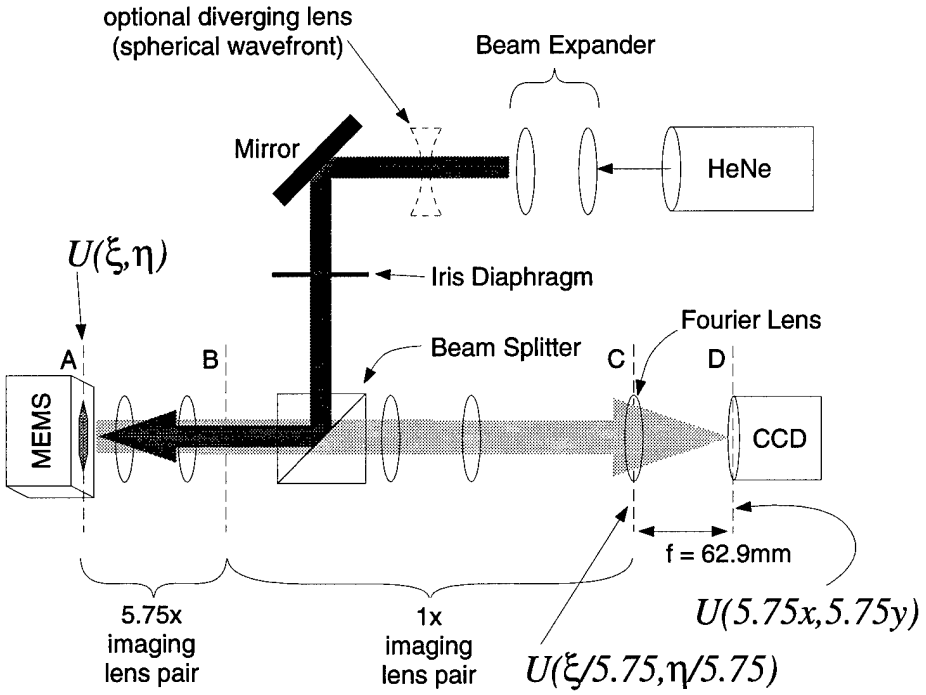

Figure 27. Laboratory setup used to measure diffraction by a MUMPS9 device.

then reproduced in plane C, using a $1\times$ lens pair. Plane C is also where the Fourier lens is placed, so the far field diffraction pattern of the virtual, enlarged MUMPS9 device can be measured one focal length further (in the back focal plane), using a CCD. Also, due to limitations of the laboratory equipment, only 64 of the 127 mirrors in the array could be controlled, so a circular iris diaphragm was used to restrict illumination to the central 61 mirrors. The normalized log diffraction pattern measured in the lab, for a plane wavefront, is shown in figure 28.

Due to the lab setup, four changes to the basic model were required:

- The effect of the circular iris diaphragm was accounted for by masking the reflectivity function, as depicted in figure 29. In reality, the aperture affects the wavefront function, not the reflectivity function. For the model, however, the two methods are equivalent.

- The 5.75× magnification of the MEMS device had to be accounted for. This was done by multiplying the object sample spacing $\Delta\xi$ by 5.75, and dividing the image sample spacing Δx by the same amount.
- Since a converging lens ($f = 62.9$mm) was used to obtain the far field pattern, $z = 62.9$mm is the value to be used to calculate the sample spacing of the sampled diffraction pattern.
- Finally, for the aberrated wavefront calculations, the curvature of the wavefront was given by its radius of curvature of 4.5m, so a new sampled wavefront function had to be created using that radius.

The predicted normalized log diffraction pattern for the lab setup, with a plane wavefront, is shown in figure 30. On closer inspection (figure 31), it can be seen that the diffraction orders are spaced and oriented the same way for both the measured and predicted patterns. The patterns appear rotated $\pi/12$ radians with respect to each other because the MUMPS9 device in the lab was aligned vertically, while the model has the device oriented $\pi/12$ radians from vertical, as described in the previous chapter.

A comparison of the radial average distribution plots is also instructive (figure 32). The diffraction orders are again shown to coincide for the measured and predicted patterns. However, the measured pattern shows more energy in the higher orders than does the predicted pattern. This can be due to other aberrations, introduced by the optical components of the lab setup, that weren't accounted for, to the limited dynamic range of the CCD, and to background lighting that was not simulated. Also, the spatial sampling of the predicted diffraction pattern was 4.58 μm, while that of the measured pattern was 15.6 μm, less than a third of the predicted pattern's. Another issue has to do with the effects of aberrations and their correction. Introducing the aberration causes a -13.66dB drop in the peak intensity of the predicted pattern, but only a -7.96dB drop in the measured case. The predicted, corrected peak intensity is down -10.80dB from the unaberrated value, but only a

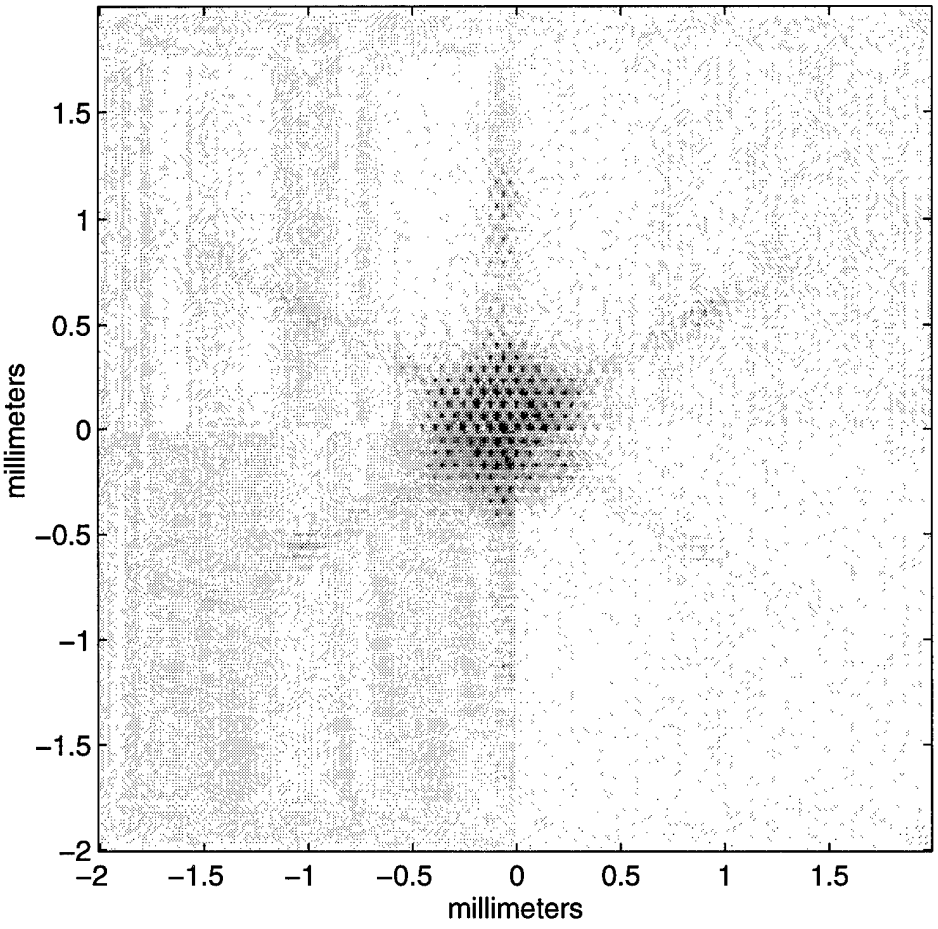

Figure 28. Normalized log diffraction pattern measured in laboratory. Because of the limited dynamic range of the CCD, this image had to be thresholded at -15dB, normalized to the peak intensity. Black pixels represent 0dB; white ones -15dB or less.

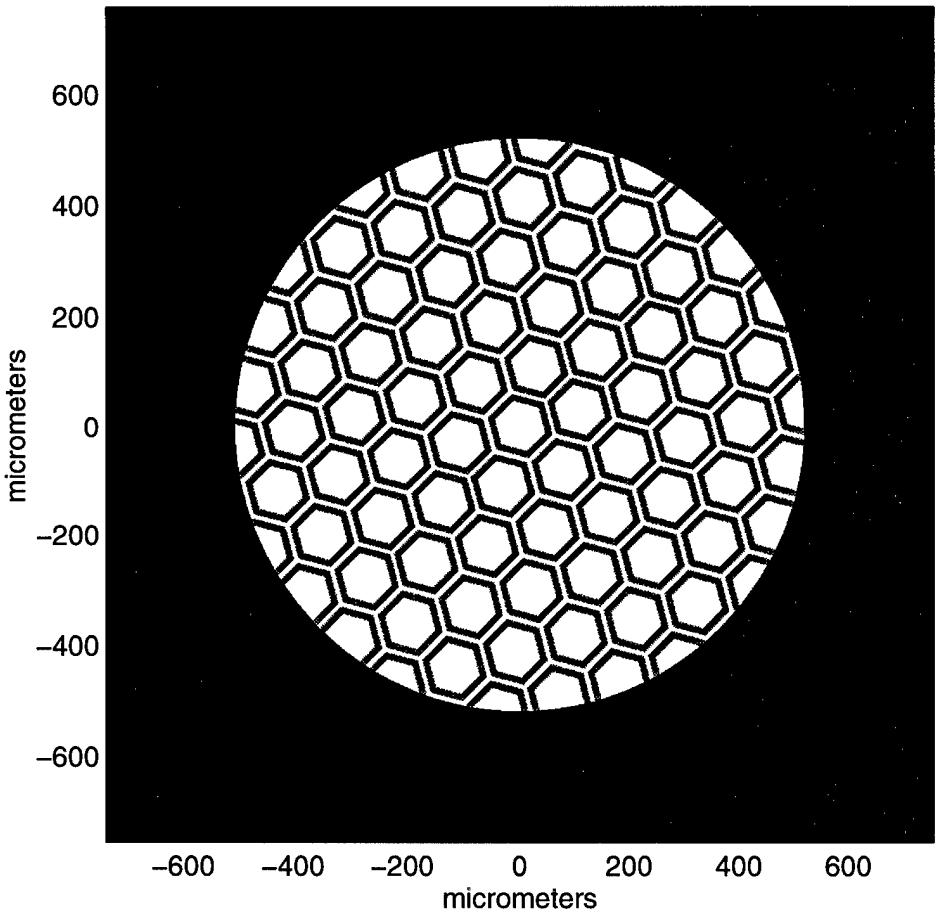

Figure 29. Masked reflectivity function used to simulate the laboratory setup.

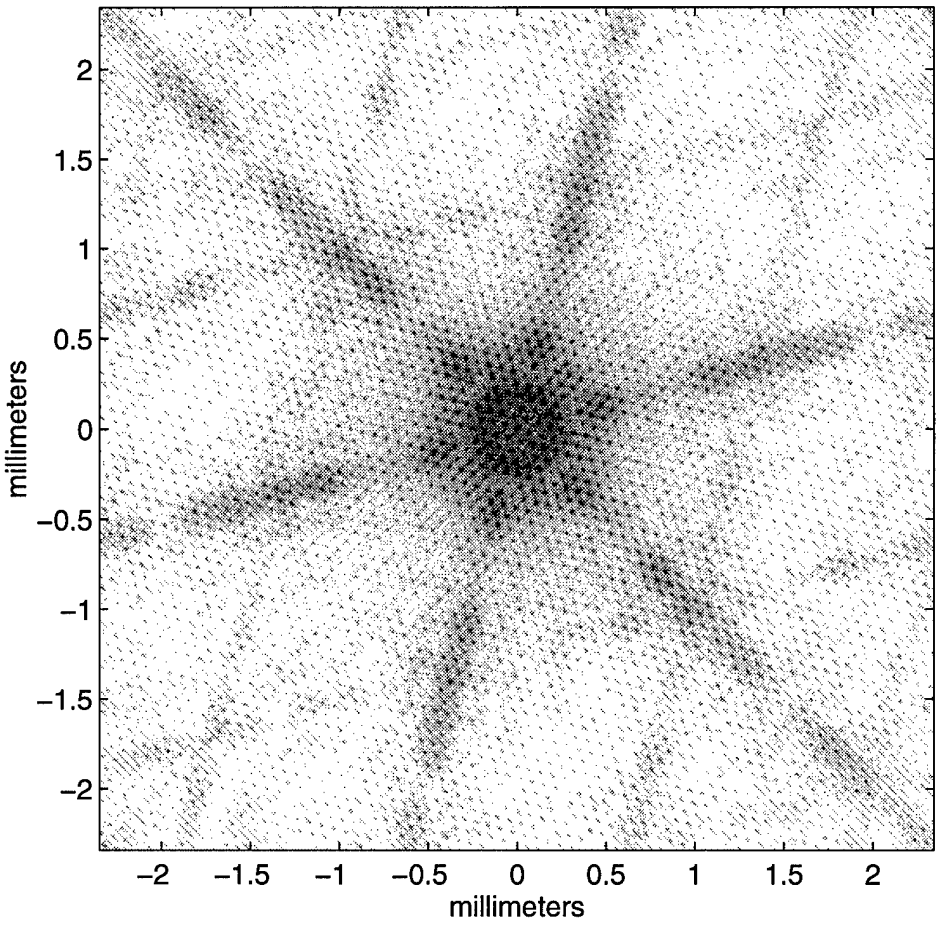

Figure 30. Normalized log diffraction pattern predicted for lab setup.

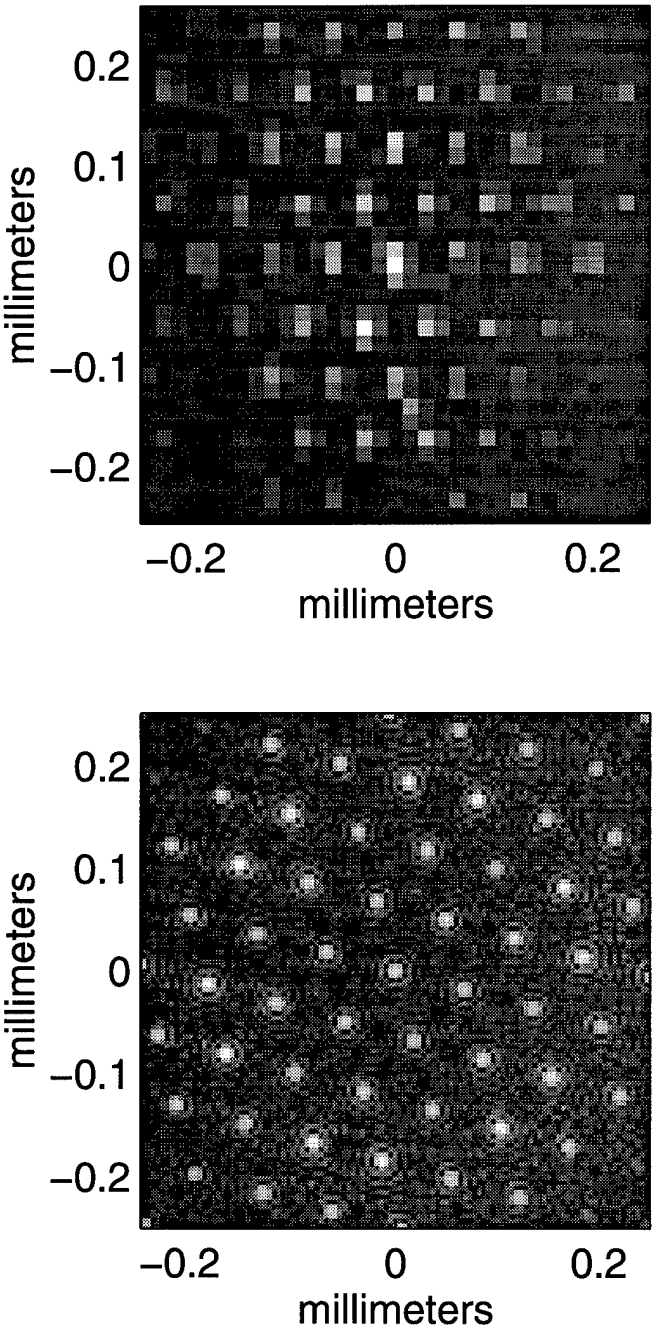

Figure 31. Close-up comparison of measured (top) and predicted (bottom) diffraction patterns for lab setup. Note: the diffraction patterns are displayed here as positive images.

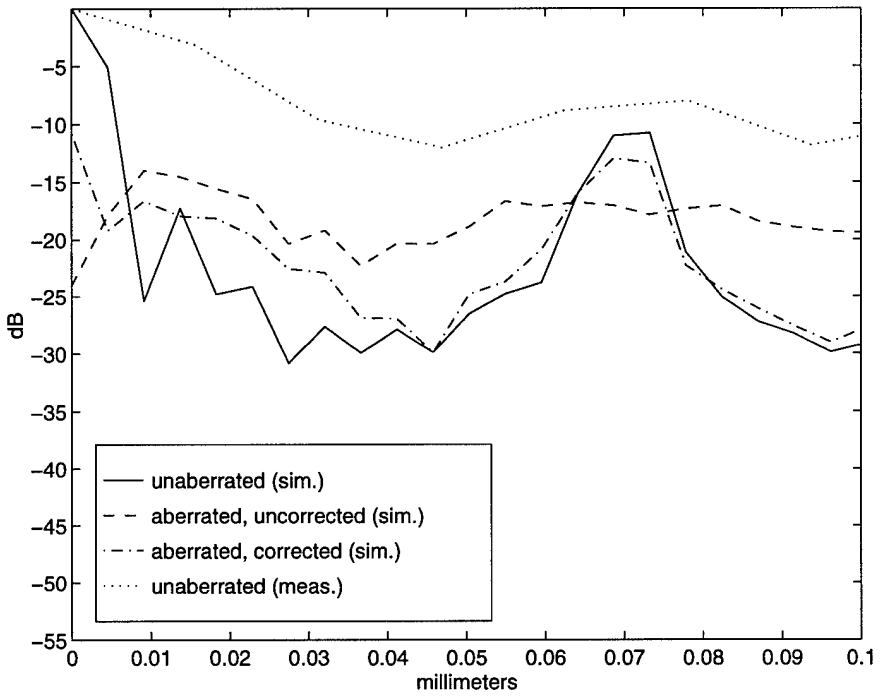

Figure 32. Normalized radial energy distribution for correction of a spherical aberration using a MUMPS9 array in the laboratory setup; comparison to measured distribution.

-6.38dB drop was measured in the lab. In order to investigate this lack of agreement between the measured and predicted values, the predicted ratios were calculated for sixty different radii of curvature of the aberrated wavefront, evenly spaced from 3.15m to 5.85m. The results (Figure 33) show a strong dependence of the corrected ratio on the amount of aberration. A small variation between the 4.5m value used to compute the predicted diffraction pattern and the actual amount of aberration would therefore be sufficient to explain the difference between the measured and predicted peak ratios.

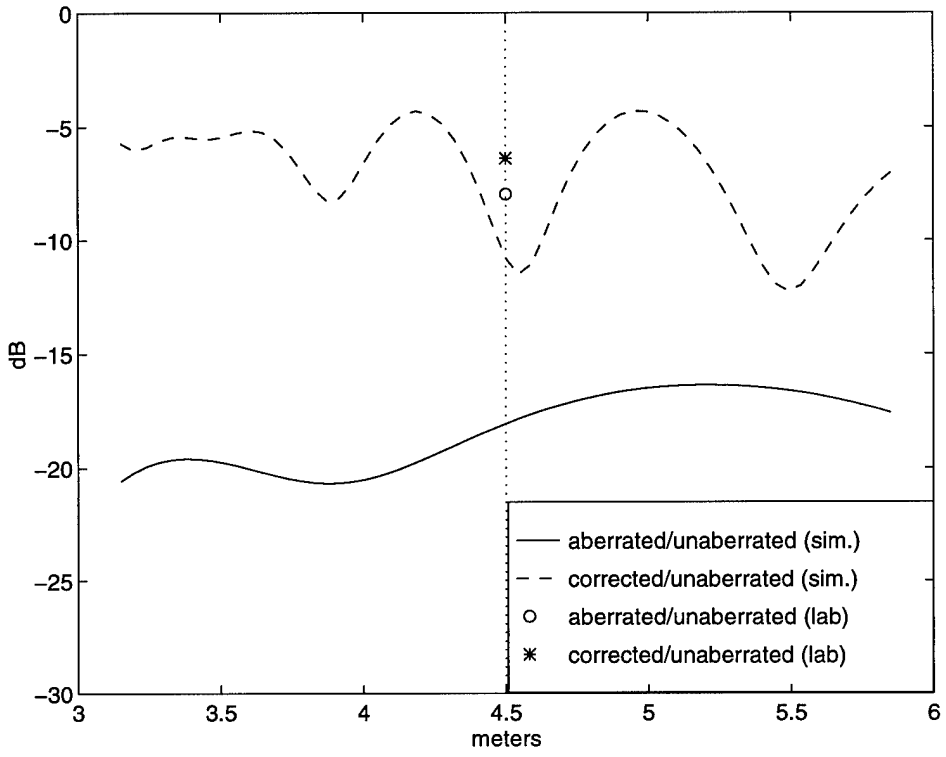

Figure 33. Peak intensity ratios as a function of radius of curvature of an incident spherically aberrated wavefront.

3.4 Lenslet Model

It is apparent that even under ideal conditions, reflection from a MUMPS9 device causes a significant amount of energy to go into sidelobes, instead of the central peak, where it is needed. This is an effect of diffraction caused by the gaps between the metallized surfaces of the MUMPS9 device. One method of eliminating this effect is to use an array of lenslets from a Hartmann wavefront sensor. These lenslets are square, 203μm to a side, with no empty space between them. The idea is to place this lenslet array one focal length above a micromirror array, which would be constructed with a mirror at the same location as each lenslet. Then the portion of wavefront passing through a given lenslet could be focused into a small spot on the underlying mirror, and on reflection would be recollimated by the same lenslet. Piston of the mirror could still be used to control the phase of that portion of the wavefront.

An appropriate mirror array does not exist at this time, but can be modeled. The net effect of displacing a mirror behind a lenslet is to change the phase of the section of wavefront leaving the lenslet. Since the lenslets are contiguous, the system is therefore equivalent to an array square micromirrors of the same dimensions as the lenslets, with no empty space between them. A model of an 8×8 lenslet array is shown in figure 34. The top image shows the reflectivity function, while the lower image shows the height function for the array when deflected to correct for a spherical aberration. The empty space around the array is zero-padding to decrease the sample spacing of the predicted diffraction pattern.

The diffraction pattern from reflection of a plane wave on the undeflected array is shown in figure 35. This is simply the pattern expected for a square aperture. Figure 36 shows the result of reflecting a spherical wavefront on the square aperture. The pattern is blurred, and the peak intensity is down to 17.34% (-7.61dB) of the peak for the plane wave's diffraction pattern. However, the corrected peak (figure 37) is back up to 88.03% (-0.55dB) of the plane wave value, 0.45dB better than

was possible with the bare MUMPS9 device. In addition, the first sidelobe of the MUMPS9 plane wavefront diffraction pattern was -15dB below the peak value, while the first sidelobe of the plane wavefront diffraction pattern for the lenslet model (figure 38) is -18dB below the peak. The difference is more dramatic for higher order lobes, with the energy dropping more rapidly with distance from the optical axis for the lenslet model than for the bare MUMPS9 model. This shows that the lenslet array is a more efficient way of controlling the phase of a wavefront, since it puts more of the energy into the central peak, instead of spreading it around like the MUMPS9 device does.

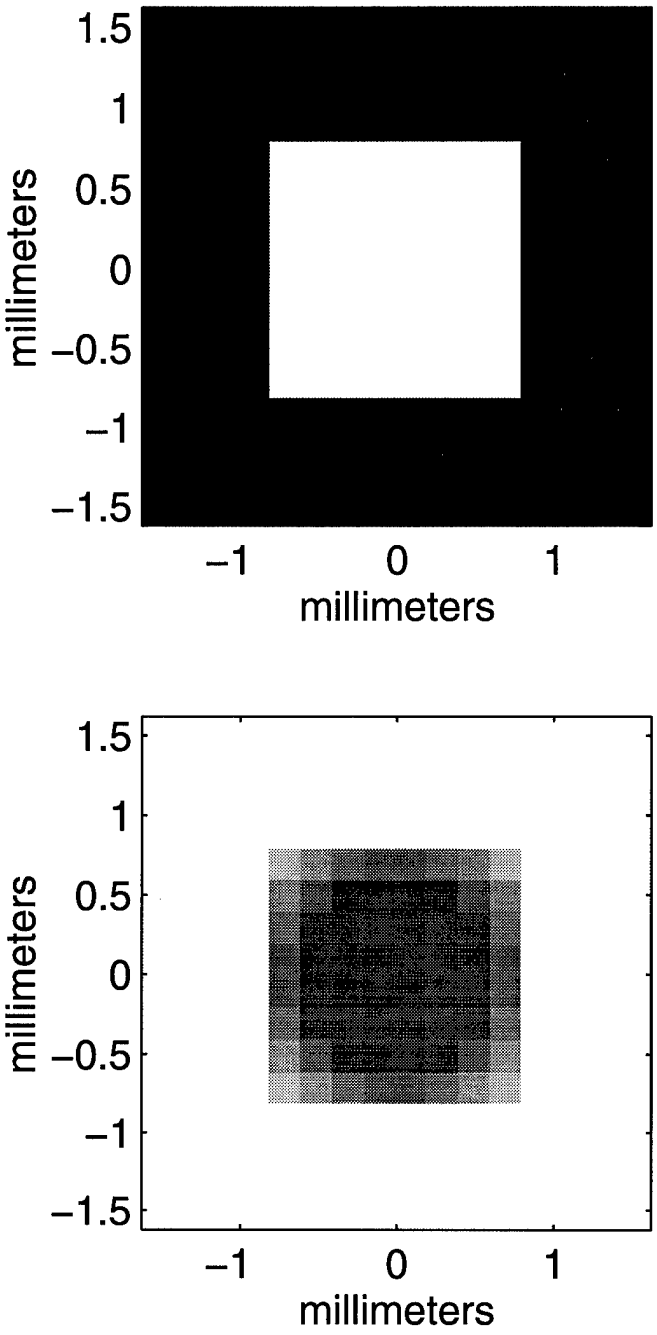

Figure 34. Reflectivity and height images for the lenslet model.

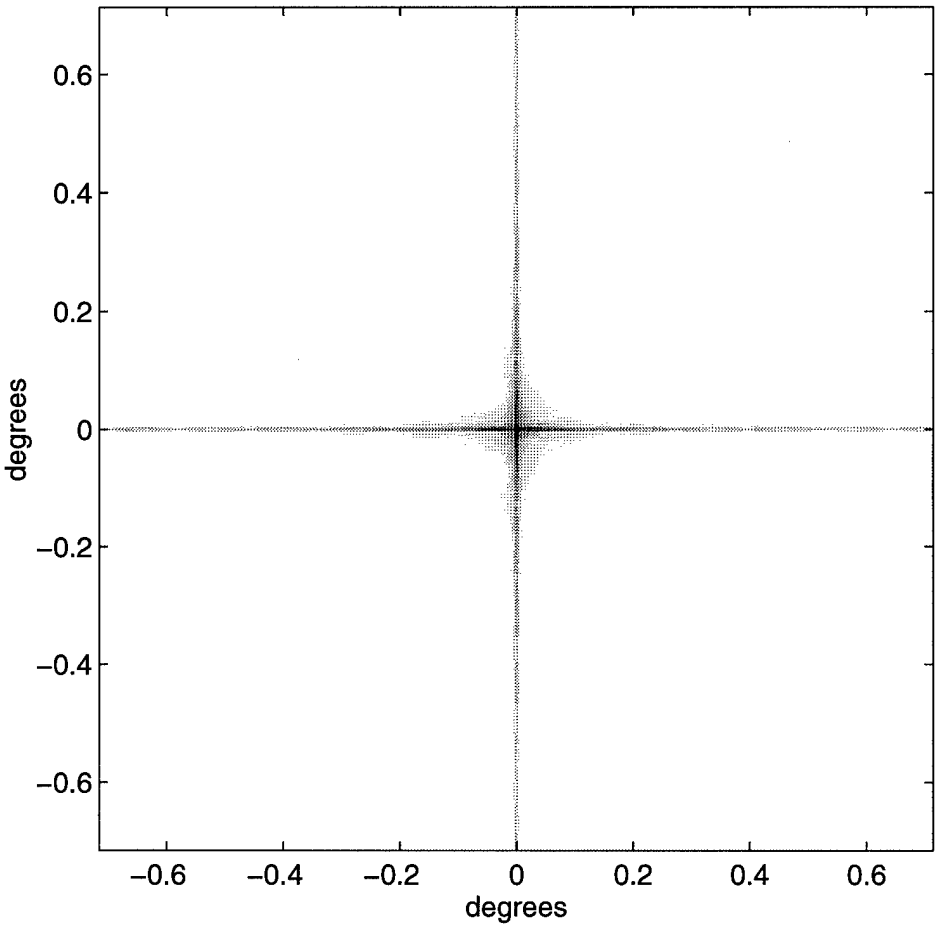

Figure 35. Diffraction pattern for a plane wave, lenslet model.

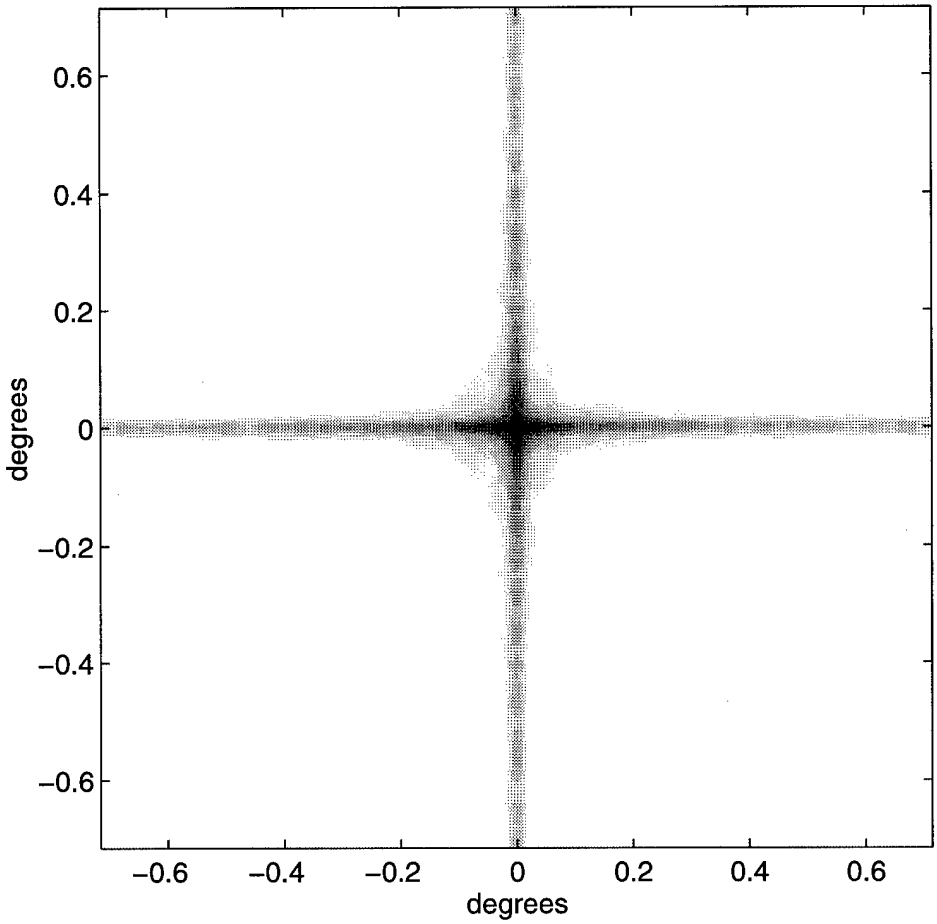

Figure 36. Diffraction pattern for a spherical wave, lenslet model.

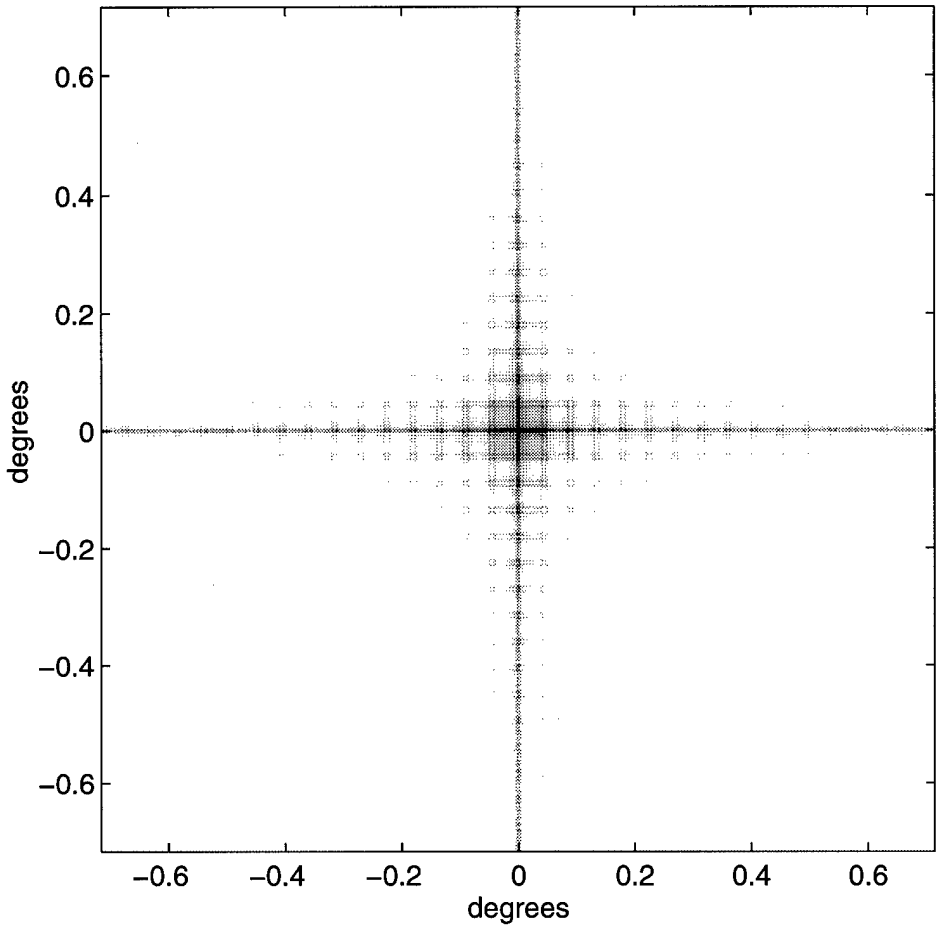

Figure 37. Diffraction pattern for a corrected spherical wave, lenslet model.

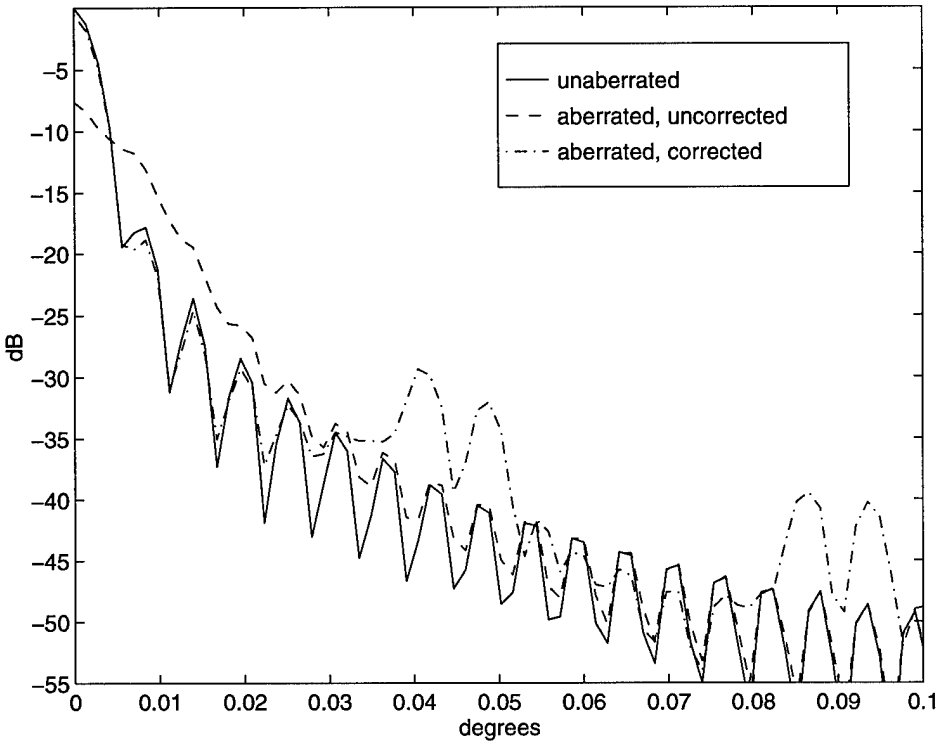

Figure 38. Normalized radial energy distribution for correction of a spherical aberration using a lenslet/micromirror array.

IV. Conclusion

Atmospheric turbulence, resulting from daily heating and cooling, distorts the shape of a wavefront traveling through it. As a result, ground-based imaging systems often cannot achieve their maximum theoretical resolution. Adaptive optics combat this effect by using a deformable mirror (DM) to correct the aberrated phase of the wavefront. The current DM of choice is a continuous flexible membrane mirror with piezoelectric actuators, which is difficult to build and maintain. A proposed alternative is an array of microelectromechanical deformable mirror (MEM-DM), consisting of an array of micromirrors built on a silicon chip, aranged to form a segmented DM. In order to shorten the design cycle of MEM-DMs, this thesis proposed a method of simulating their optical characteristics, given a physical description of the design to be modeled.

The basic idea behind simulating diffraction by a MEM-DM was to create a sampled version of the optical field after reflection which takes into account the reflectivity and geometry of the device, as well as the aberration of the incoming wavefront, and have a computer produce its discrete Fourier transform, which represents the far-field diffraction pattern. Since an optical disturbance has both amplitude and phase, each sample point was complex-valued, with the absolute value representing the reflectivity of the device at that point, and the phase representing a combination of the phase of the aberration and the phase change induced by the variable height profile of the device.

The simulation shows that MEM-DMs can be used to correct a wavefront for aberrations. For a spherical aberration with a total phase variation of 2π across the surface of the device, the peak intensity of the aberrated output can theoretically be corrected to within 1dB of the peak intensity of the unaberrated output. Furthermore, the simulation was able to predict the diffraction pattern that was obtained with a given laboratory setup. Correction of a spherically aberrated wavefront, using

an actual MEMS device, was also demonstrated by Hick (15). Both the simulation and laboratory results show that the current state of the art MEMS micromirror devices diffract a significant fraction of the incoming energy into sidelobes, reducing their efficiency as deformable mirrors.

In order to be useful in shortening the design cycle for MEMS devices, the simulation was designed to be easily adaptable, so that it could be used to model different types of MEMS designs before they are even produced. A new design for MEMS deformable mirror device involves aligning the mirror segments with an array of lenslets, the idea being that the incoming wavefront will be focused onto the centers of the micromirrors by the lenslets, then recollimated as it leaves the system through the lenslet array. The main benefit would be to mostly bypass the diffraction effects of the underlying MEMS device. Such a system was modeled, and was found to be significantly more efficient than the current MEMS device in correcting an aberrated wavefront, with the peak intensity being returned to within 0.55dB of its unaberrated value, and with much less energy being lost to sidelobes.

Appendix A. Running the MEMS Optical Modeler

A.1 Model Design

A.1.1 The modeling process. The modeling process involves the following steps:

1. Create a sampled reflectivity function, $R(\xi, \eta)$ representing the device being modeled. Each sampled point will have the complex value $R = re^{i2\pi(2h/\lambda)}$, where r represents the absolute reflectivity at that point and h is the height of that point above a reference plane.
2. Create a sampled wavefront function $W(\xi, \eta)$ that has the same resolution as R.
3. Define the optical disturbance in the source plane: $U(\xi, \eta) = W(\xi, \eta) \times R(\xi, \eta)$.
4. Have the computer calculate $|\mathcal{F}\{U(\xi, \eta)\}|^2$ and normalize it. This is the normalized far field diffraction intensity pattern.
5. Use equations 10 and 6 to determine the scale of the predicted pattern.

A.1.2 Data structures. The software to create the MEMS optical model is based on Matlab^{tm}, a commercial mathematics program. The basic data item in Matlab is the matrix, or two-dimensional array. This structure is, quite conveniently, also useful for representing digitized images, which are simply two-dimensional arrays of picture elements, or pixels. To avoid having three different words—matrix, array and image—tied up to describe the same thing, the following convention will apply in this appendix:

"Matrix" is the term used in Matlab manuals to denote Matlab's basic data item.

"Array" refers to an array of micromirrors. For example, the MUMPS9 device is an array of 127 micromirrors.

"Image" is what will be used to refer to any Matlab data item that contains a graphical representation of a MEMS device. For example, the reflectivity function $R(\xi, \eta)$ of the device will be modeled as a "reflectivity image". Only square images will be used (same number of rows and columns).

A.1.3 Data flow. Matlab allows user-defined scripts and functions, and the model will be implemented as a collection of these. The only file in the model that contains actual MEMS device data is called the device generating function, or `genfunc`. Of course, the actual file can have any legal file name. It is the only one that needs to be created or modified by the user to model a different MEMS device. The actual model is created using a collection of Matlab functions, which call on `genfunc` to provide data about the device being modeled.

The ability to build the model one step at a time combined with the built-in flexibility of the `genfunc` should allow the user maximum opportunity to fine-tune the model as it is being built. Also, creating a computer image of a real device involves frequent changes of coordinate systems, from physical coordinates to pixel coordinates and back. The writer of the `genfunc` only needs to worry about the physical coordinates, as the other functions take care of doing all the transformations to pixel coordinates. Appendix B covers the finer points of creating a `genfunc`.

A.2 Construction of model

This and subsequent sections cover the steps used to create a MEMS optical model. For illustration, the result of executing each step with `hex2`, the `genfunc` for the MUMPS9 hexagonal array, will also be shown.

A.2.1 Modeling an individual micromirror. To take advantage of the symmetry in an array of micromirrors, the first step in creating the model is to make an image of an individual micromirror. When writing the `genfunc`, the individual mirror is first broken down into areas of common height and reflectivity (for exam-

ple, all the Poly2 surfaces). Each of these areas is assigned an index number and is then further broken down into a set of convex polygons. The polygons, in turn, are defined as a set of lines and one "seed" point that is inside the polygon. The seed points are used by a low-level fill routine that sets all pixels in the polygon to the appropriate index number. Next, each line is defined by its end points. Finally, the end points are defined simply by their coordinates.

So to define the image of an individual micromirror, three data items are needed:

- A list of points, (x, y), in real coordinates (as opposed to pixel coordinates),
- A list of lines, $(pt1, pt2, area\#)$, where $pt1$ is the index, within the list of points, of the line's first end point, and $area\#$ is the index number of the area that the line borders, and
- A list of seed points $(x, y, area\#)$.

To actually draw the image, two more data items are required, to set the resolution of the image: the physical extent of the mirror, programmed into the `genfunc`, and the number of pixels desired in the image. The latter is entered by the user when calling `mam2`, the function that Makes A Mirror.

The function `mam2` takes two arguments: the desired size of the image `len`, and the name of the `genfunc` to use. Creating an image of an individual MUMPS9 micromirror, with a resolution of 270×270 pixels, is accomplished with the following Matlab command:

```
>> mirror = mam2(270,'hex2',spec);
```

The last argument is the `spec` referred to in the previous section; it is not used by `hex2` when making an image of an individual mirror, so it doesn't matter what value it is given. When `mam2` is given the name `hex2`, it knows to make the following calls:

- `hex2(1,spec)` returns the list of vertices in the image,

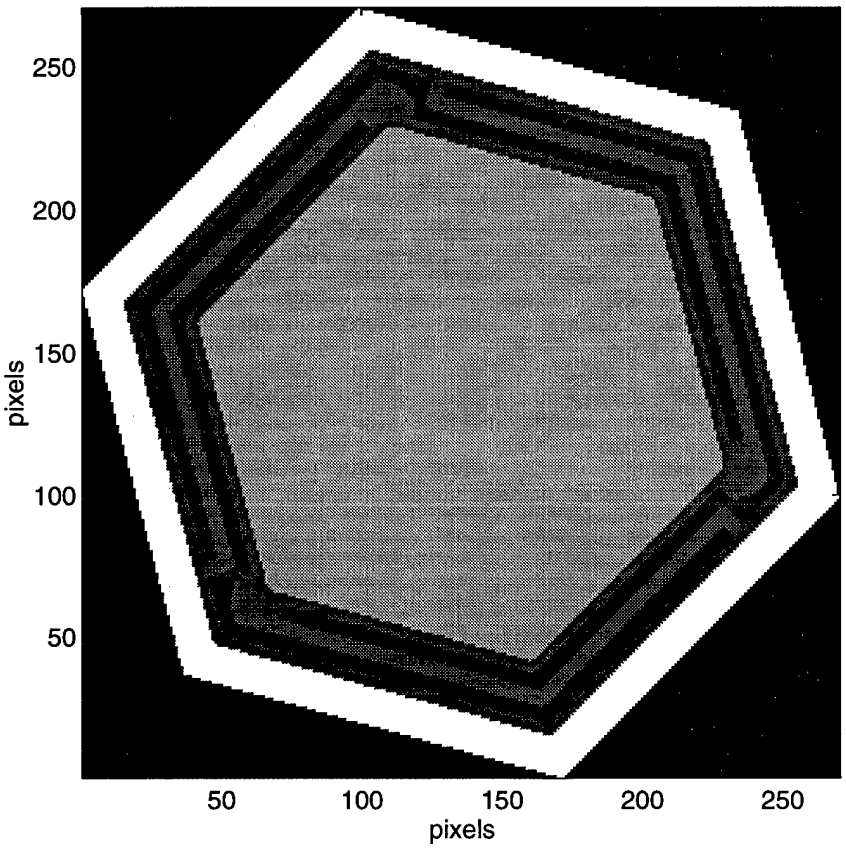

Figure 39. MUMPS9 micromirror indexed image.

- `hex2(2,spec)` returns the list of lines, as described above,
- `hex2(3,spec)` returns the list of seed points, and
- `hex2(4,spec)` returns the physical extent of the image.

The output of the above command is shown in figure 39.

Actually, the image of an individual mirror needs to be 90 × 90 pixels to properly fill up a 1024 × 1024 pixel image when tiled into an array, but that would be too small for the filling routine, `pfilll` to fill in all the polygons (the lines would be too close together for `pfilll` to work properly), so the image was generated at three times the required size, then downsampled with the command

```
>> mirror = mirror(1:3:270,1:3:270);
```

A.2.2 Tiling micromirrors into an array. The next step is to tile the image of the mirror into an array. For this, some additional information is required. First, the desired size of the final image must be given by the user. This will be the size of the final sampled complex reflectivity function, of which the FFT will be computed. Since FFT algorithms are most efficient when the number of samples is a power of 2, the size of the array image is restricted to be a power of 2, and the actual argument passed is the power desired. Also required are the locations of the central points of each mirror in the array; these are obtained from the `genfunc`. To create a 1024 $\times$ 1024 pixel image of a MUMPS9 array, the Matlab command is

```
>> marray = mama2(10, mirror,'genfunc',6);
```

The output of the above command is shown in figure 40.

To make figure 40, `mama2` made the following calls:

- `hex2(7,6)` returns the list of physical coordinates of the centers of the mirrors, with the second argument specifying that there will be 6 hexagonal rings of micromirrors around the central mirror,
- `hex2(4,spec)` returns the physical extent of the image of an individual mirror. `mama2` uses this to determine the resolution of the image, and to convert the phsyical coordinates of the mirrors to pixel coordinates.

It should be noted that the resolution of the image is determined by the number of pixels in the image of the individual mirror. If the `power` argument to `mama2` is too small, or the size of the mirror image is too big, then the mirror array will overflow past the edge of the array image. Trial and error is required to determine the proper image sizes to make the array fill the image without overflowing.

Another item of interest is that since the spacing of the mirrors in the array image is determined by the `genfunc`, it is a good idea to make the outer edge of the image of the individual mirror slightly larger than reality, so that neighboring

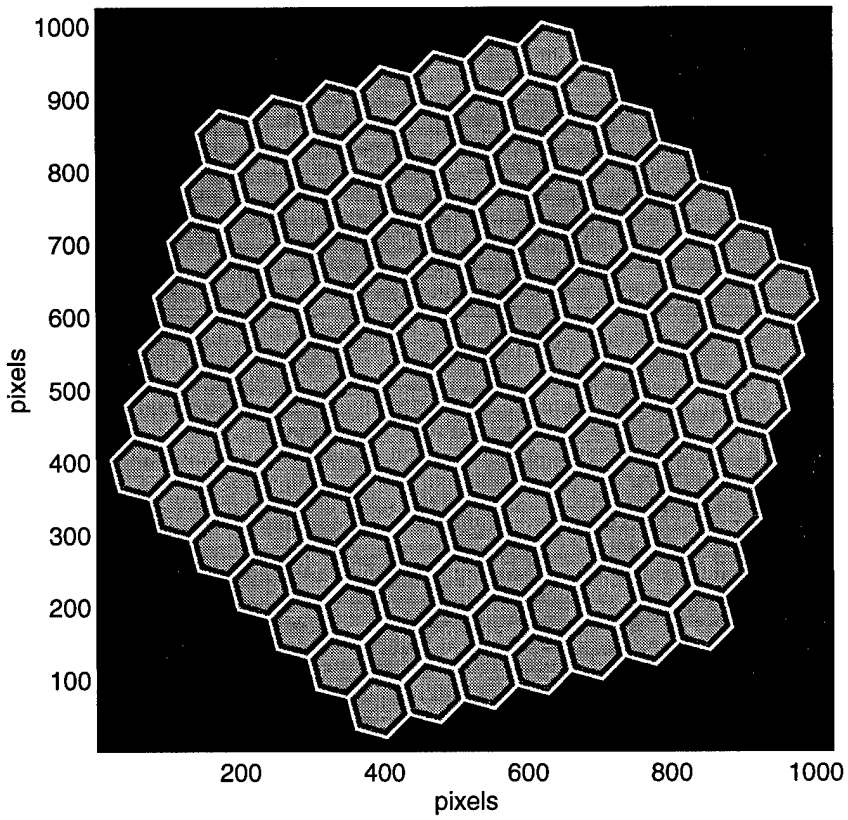

Figure 40. MUMPS9 micromirror array indexed image.

mirror images will overlap slightly in the tiled array image. This prevents blank pixels occurring between mirrors in the array image. The function `mama2` is written to correct for overlapping mirrors, as long as the part of the image that overlaps has the highest *area#*. In `hex2`, for example, the metallized band between each mirror is 10 individual mirror is drawn with a 6 μm wide border of metallized area, instead of 5 μm.

A.2.3 Creating height and reflectivity images. So far we have created only indexed images. The values of the pixels in figures 7 and 40 represent only which surface that pixel belongs to. What is needed are images of the reflectivities and heights. Fortunately, `genfunc` provides the information needed to cross-reference an *area#* to either a reflectivity or a height. The functions that pull the information out and apply it to the image are `mhm2` for height and `mrm2` for reflectivity. The Matlab command

```
>> hiray = mhm2(marray,'hex2',spec);
```

produces the image in figure 41, and the Matlab command

```
>> refray = mrm2(marray,'hex2',spec);
```

produces the image in figure 42.

A.3 Application of control inputs

The next step in the development of the MEMS model is to simulate control inputs that cause the individual mirrors to move downward. A set of control routines was created to do this.

A.3.1 Control routines. These are independent of the `genfunc`, because they are intended to work on images of arrays that have already been constructed. More specifically, they operate on the height image created in the previous section using `mhm2`.

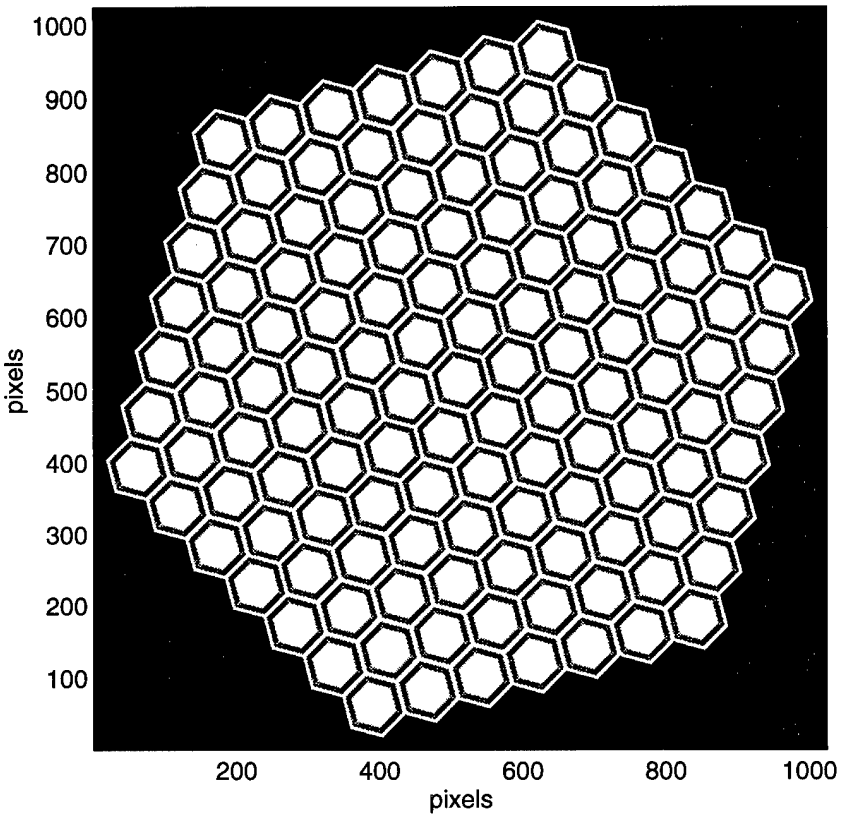

Figure 41. MUMPS9 micromirror array height image.

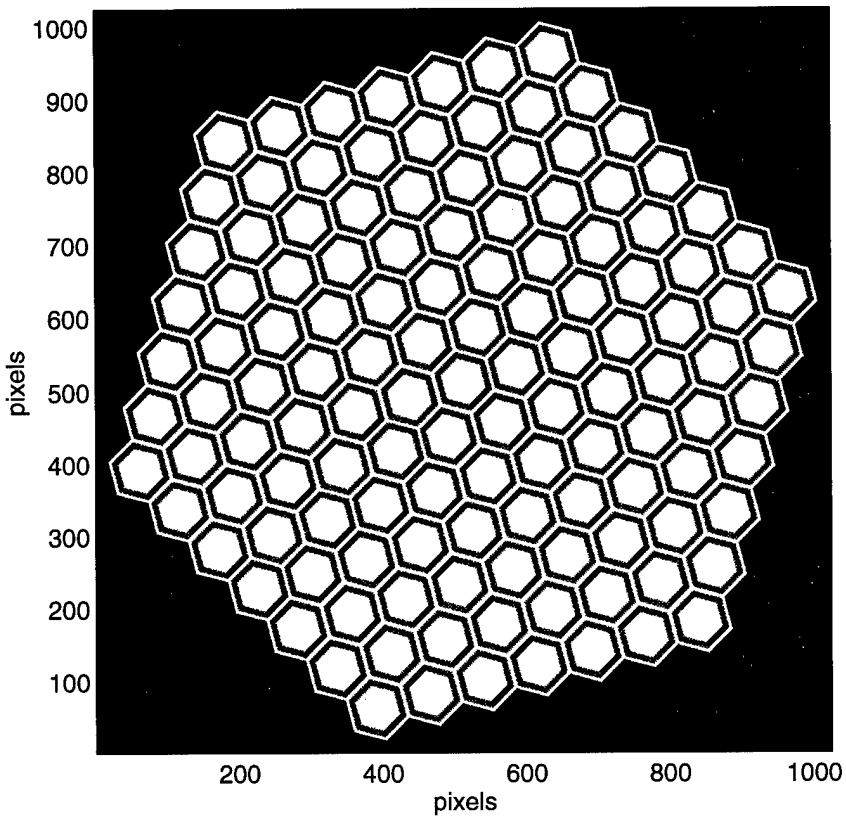

Figure 42. MUMPS9 micromirror array reflectivity image.

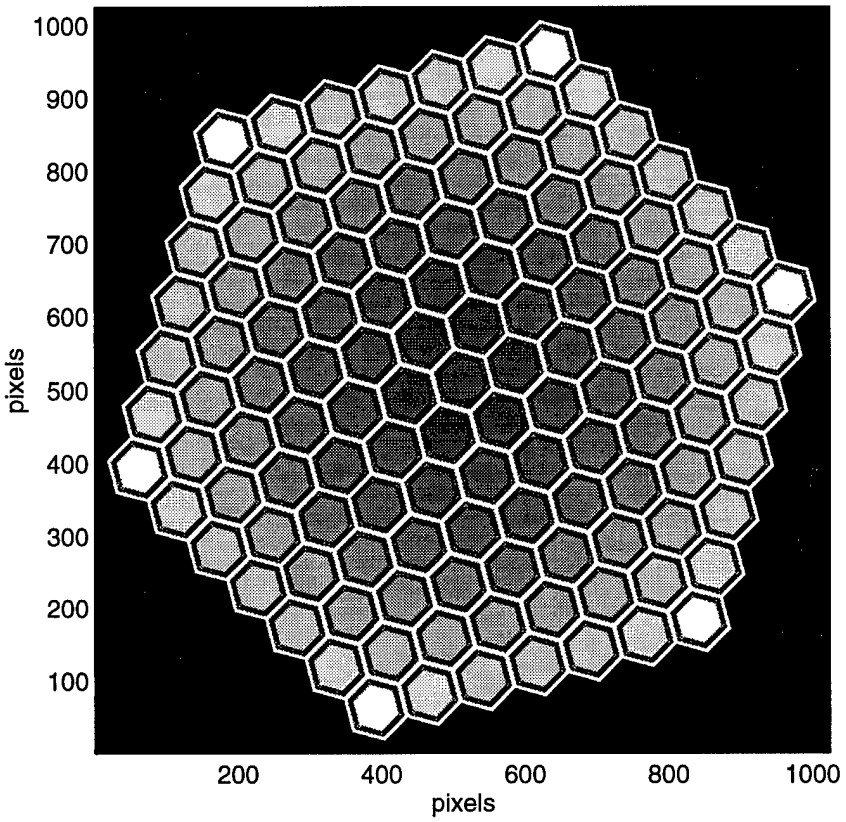

Figure 43. MUMPS9 micromirror array height image, parabolic profile.

Their basic method of operation is to scan over each pixel in a look-up image (described below), to find the pixels that need to be modified. When such a pixel is found, the routine goes to a look-up table (also below) to determine the location of the center of the mirror to which the pixel belongs. This is what is used to calculate the amount of deflection of that pixel, since all pixels on any given mirror must move in unison. The required deflection is then modified by the value in a responsiveness image (below). If the resulting deflection is more than one wavelength, then it is adjusted again, so that it lies between 0 and 1 wavelength. As an example, the result of "cupping" a MUMPS9 array to give it a parabolic profile is shown in figure 43. The command used to generate the image was

```
>> pararay = parabolize2(1,0.6328,hiray,conray,respray,table);
```

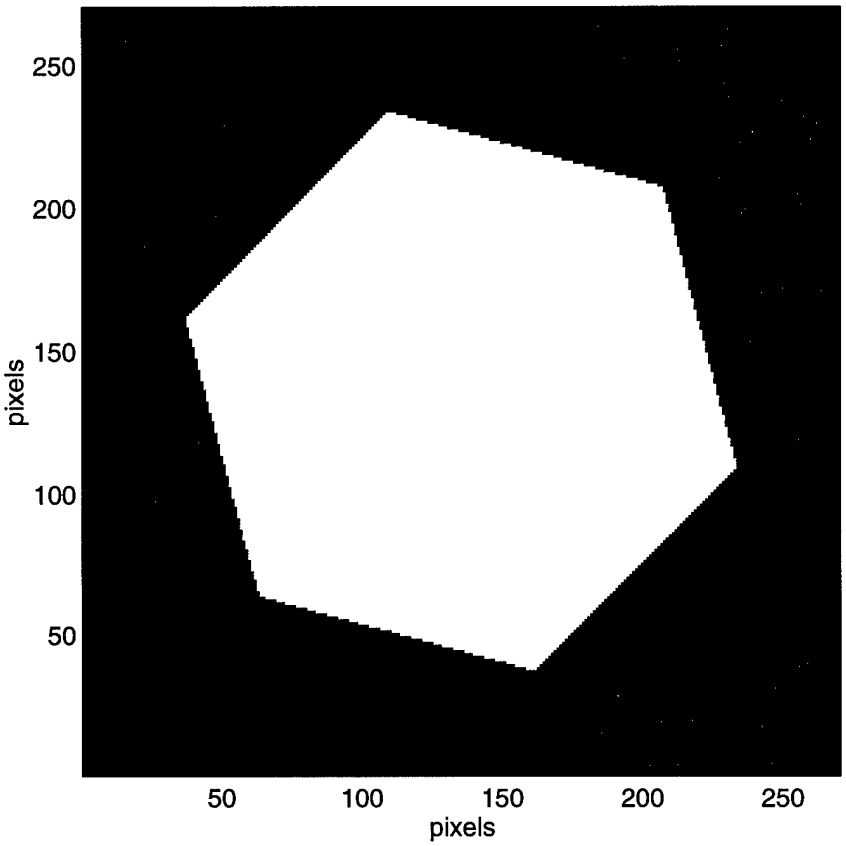

Figure 44. MUMPS9 micromirror control surface image.

A.3.2 Control surfaces. When subjected to a control input voltage, one part of a micromirror is deflected, while the rest stays put. In order for the control functions to know which pixels move and which don't, it is necessary to make an image of the portion of the array that can move. Just like for the array image, this is done by making an image of the control surface of an individual mirror and then tiling it. The function `macs2` makes an image of the control surface of an individual micromirror. The image is constructed in the same way as the image of an individual mirror. However, there are no *area#*s. Instead, the pixels are assigned 0 where they don't move, and 1 where they do. The control surface for a MUMPS9 micromirror is shown in figure 44. The movable portion corresponds to the metallized portion and the Poly2 edge of the mirror.

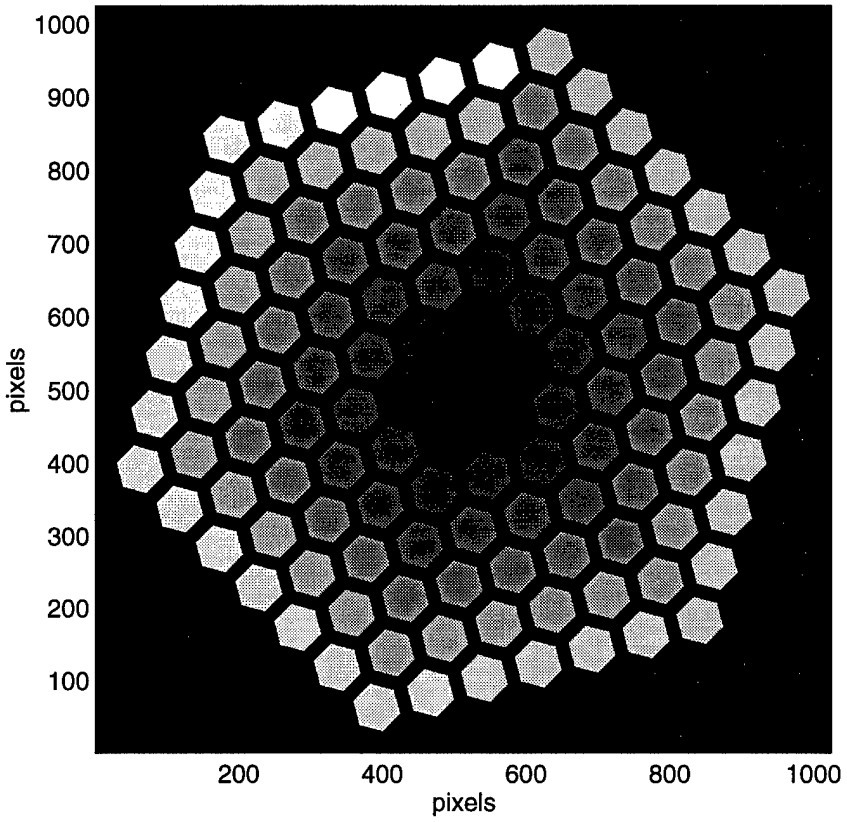

Figure 45. MUMPS9 micromirror control surface array image.

A.3.3 The look-up image. Things become even more interesting when it comes to tiling the images of control surfaces. The routine to use is `macsa2`, and it produces an image of the array of control surfaces. The image is indexed with each mirror getting a unique index number. This is the array that the control functions scan to find pixels to move. Pixels that have a value of 0 represent points that do not move. Pixels that do move will have a value equal to the index number of the mirror to which they belong. The control function then takes this value, and goes to the look-up table to find the location of the center of the mirror, and uses this to compute the amount of deflection. The look-up image for a MUMPS9 array is shown in figure 45.

A.3.4 The look-up table. The function `malut2` returns the look-up table used by the control functions. It takes the same arguments as `mama2`. The table cross- references the index numbers in the look-up image with the locations of the centers of the mirrors.

A.3.5 The responsiveness image. This is a semi-implemented feature of the modeler. Since the flexures are attached to a moving mirror at one end, and a fixed frame at the other, it is obvious that the height of a pixel on a flexure would move by an amount that depends not only on the amount of input voltage, but also on its location along the flexure. The responsiveness image is intended to allow this to be modeled. The value of a pixel would indicate the extent of the effect of the control input on its height. The pixels of the mirror would have value 1; pixels that do not move at all would have value 0; and pixels on the flexures would have intermediate values depending on their location. Once the total amount of deflection required for a given mirror is known, the height of each pixel is decreased by that amount multiplied by its responsiveness value.

Unfortunately, this feature turned out to be rather complicated to implement. Given the small size and low reflectivity of the flexures, it was determined that the impact of not modeling the bending of the flexures would be small. However, the use of a responsiveness image is implemented in the control functions, so it is necessary to make a "dummy" responsiveness image by tiling the individual control surface image with `mama2`. This makes the mirrors and the Poly2 edges undergo the entire deflection, while everything else, including the flexures, stays put.

A.4 Calculation of diffraction patterns

A.4.1 Modeling a wavefront. The wavefront that is to be reflected from the MEMS device is much easier to model. The only requirement is that it have the same resolution and size as the device image. For a plane wave, the value of

each pixel would be 1, so the Matlab function **ones** can be used. Tilted or spherical wavefronts, on the other hand, involve a phase that varies across the image. The commands

```
>> wave = masw(deflection,deltax,power)
```

and

```
>> wave = maswr(radius,deltax,power)
```

are used to create an image of a spherical wavefront, depending on whether the curvature of the wavefront is defined by the total deflection at the center of the image (`masw`) or by its radius (`maswr`).

A.4.2 Computing the far-field pattern. The Matlab command to calculate a diffraction pattern is

```
>> diff = diffract(wavefront,wavelength,refray,hiray);
```

This takes the reflectivity and height images, combines them into a complex reflectivity image and multiplies the result with the wavefront image. The result is a sampled representaion of the source optical disturbance $U(\xi, \eta)$. It then performs a two-dimensional FFT to predict the far-field diffraction pattern. The intensity of the diffraction pattern is given by

```
>> diffint = abs(diff).^2
```

The diffraction intensity pattern for an undeflected MUMPS9 array is shown in figure 46.

A.4.3 Computing the scale. All the figures so far have been displayed with the axes labeled in pixel coordinates. It is more instructive to label the axes with their physical values, so the function **getaxes** was created to do this. It returns two one-dimensional data arrays. The first one contains the values corresponding to the

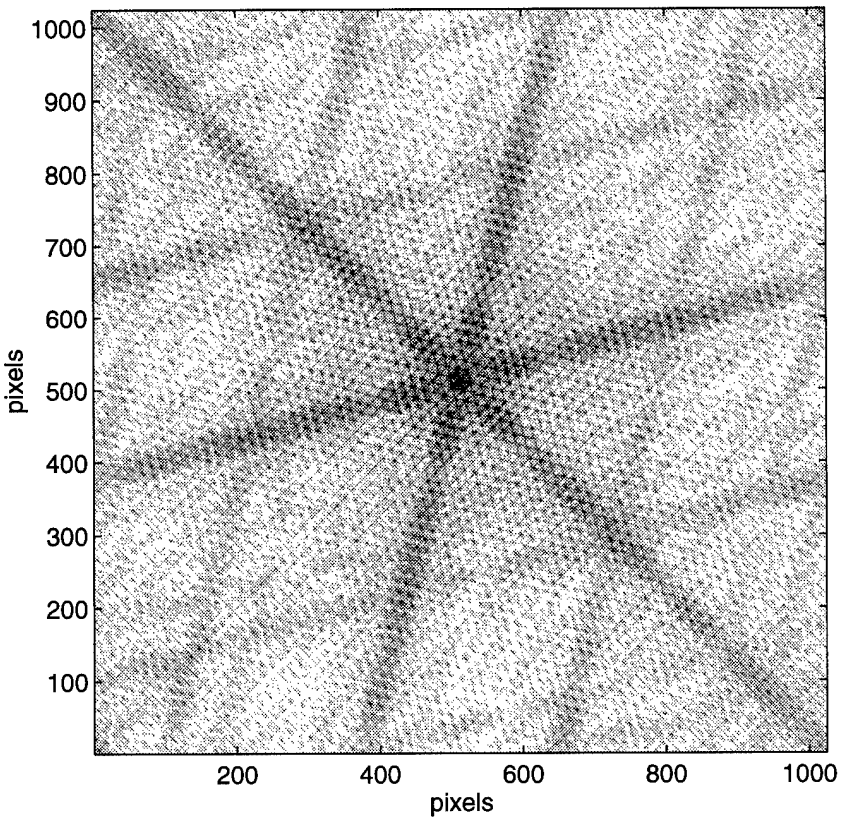

Figure 46. Simulated normalized log far-field diffraction intensity pattern for a plane wave reflected from an undeflected MUMPS9 micromirror array. The pattern is displayed as a negative image, thresholded at -90dB from the peak intensity.

physical location of each row or column in the device images (ξ or η), and is in the same units that are used in the `genfunc` to describe the device. For example, `hex2` describes everything in μm. The second array returned by `getaxes` contains the associated spatial frequency values $(\omega_\xi, \omega_\eta)$ for the diffraction pattern. These are in inverse units of length, and must be multiplied by λz to get units of length, or by λ to obtain the angular spectrum. The command to obtain these is

```
>> [xi,omega] = getaxes(power,genfunc,spec,table);
```

where `power` and `spec` should have the same values that were passed to `mama2` to create the array image (eg. `power = 10` and `spec = 6`).

A.4.4 Displaying images. Matlab has an extensive assortment of display functions, each with many options. For this modeling package, two additional display functions were created that use all the appropriate built-in functions and options to correctly display two-dimensional device images. The first, `show(image)`, displays `image` as a normalized, 256 level gray scale, 1:1 aspect ratio image with the axes labeled in pixel coordinates. The second, `show2(x,y,image)`, does the same thing with `x` and `y` as the axis labels. All of the images shown so far have been created with `show`.

Appendix B. Making a device function

The heart of the MEMS Optical Modeler is the "device function", a Matlab `.m`-file that contains all the information necessary for the other functions of the model to create the images described in Appendix A. While basic knowledge of Matlab is necessary in order to use the modeling functions, the device function was designed to be relatively easy to create. The device function is the single file that needs to be rewritten or modified in order to create a model of a new device. Also, since the function can be given any legal file name (as long as it has the `.m` extension), there is no limit to the number of different devices that can be modeled. Two device functions are included with this thesis:

`hex2.m` is used to model a MUMPS9 hexagonal micromirror array, and

`lens2.m` is used to model a cartesian lenslet array aligned with an equally spaced array of micromirrors.

In this appendix, any references to the above functions indicate a reference to that specific function/model, while the term "`genfunc.m`" will be used to refer to a generic device function.

B.1 The Arguments

The device function is called by the other modeling functions with the command

```
>> genfunc(rqst,spec);
```

This section discusses the significance of the two arguments.

B.1.1 `rqst`*: The Request.* This argument simply tells the `genfunc` which data item is being requested, in accordance with Table 4. The `genfunc` should

rqst	Requested Data
1	Micromirror Vertices
2	Micromirror Lines
3	Micromirror Seed Points
4	Physical Extent of Micromirror Image
5	Reflectivity Table
6	Height Table
7	Array Lattice Points
8	Control Surface Vertices
9	Control Surface Lines
10	Control Surface Seed Points

Table 4. `rqst` Cross-reference table.

therefore be written to return the appropriate data item, based on the value passed to it in the `rqst` argument.

B.1.2 `spec`*: The Specifier.* This argument is designed to provide additional flexibility to the modeler. The `genfunc` does not have to be written to take advantage of it. When using a `genfunc` that does not use the `spec` argument, however, it must still accept a second argument, since the other modeling functions will always call it with two arguments. It that case, it is simply irrelevant what the value of that argument is.

When the `spec` argument is used, its significance can be whatever the writer of the `genfunc` wants it to be, as long as it is associated with the value of the `rqst` argument that it accompanies. For illustration, both `hex2.m` and `lens2.m` use the `spec` argument in conjunction with the request for the coordinates of the array lattice points (the coordinates of the centers of the mirrors in the array), which corresponds to `rqst = 7`.

In the case of `hex2.m`, it was useful (for debugging purposes) to create hexagonal arrays with different numbers of mirrors, so `spec` is used to define the number of hexagonal rings. The command

```
>> hex2(7,1);
```

returns the coordinates of the centers of seven micromirrors: the central one and the six that are immediately adjacent to it. Since the entire array has six hexagonal rings of micromirrors, to construct it the above command needs to be issued with the second argument set to 6.

The lenslet array, on the other hand, is cartesian, so it is best described by the number of rows and columns of lenslets. Since the lenslet model is purely theoretical for now, it is useful to be able to specify the number of rows and columns desired at the time the model is built, without having to rewrite a new `genfunc` just to change those two parameters. For this reason, `lens2.m` was written so that when the command

```
>> lens2(7,spec);
```

is given, `lens2.m` expects the `spec` variable to be a two-component array `[m n]` where `m` and `n` are the numbers of rows and columns desired.

The above are the only uses of the `spec` variable implemented in `hex2.m` and `lens2.m`, but others are possible. For example, the `genfunc` can be written to take wavelength into account. Since the reflectivity (`rqst = 5`) of a material varies with wavelength, the `genfunc` can be written so that the data returned by

```
>> genfunc(5,lambda);
```

depends on the value of `lambda`.

B.2 The Output

There are ten data items that the **genfunc** needs to be able to produce on demand. To ease the creation of a device function for a new device, the model is designed so that the **genfunc** only needs to return data that is expressed in physical values, using units of length, for example. The conversion to pixel coordinates is done by other parts of the model and is, for the most part, transparent to the user. Also, since the **genfunc** is defined as a Matlab function, it is written as a series of Matlab commands. This means that while it can be written with all the data simply typed in the appropriate format, it can also contain formulas that calculate the data on the fly. The latter method was used heavily in writing `hex2.m`, to take advantage of the symmetries of the MUMPS9 hexagonal array.

B.2.1 Preliminary Considerations. The model of the device is created, ultimately, by low-level graphics functions that set pixels to given values, draw lines, etc. The device itself must be broken down into elements that can be drawn using these functions. First, areas of common height and reflectivity are defined, and assigned consecutive integer index numbers. For example, the exposed polysilicon surfaces of the MUMPS9 device are all assigned an index number of 2 in the function `hex2.m`. The area that borders the micromirror image, and that will end up adjacent to the same area of neighboring micromirrors when the tiled image is formed, must have the highest index number. Next, each of these areas is further broken down into a collection of convex polygons, defined by the lines that border them, and by a "seed point" that is any point inside it. The lines themselves are defined by their endpoints. Finally, the endpoints are defined by their x and y coordinates. Drawing the device consists of drawing all the lines, and then filling the resulting polygons with the appropriate values using the seed points. This process is discussed in more detail in Chapter 1.

B.2.2 Micromirror Vertices (`rqst = 1`*).* If the `genfunc` is asked for the coordinates of the vertices of an image of an individual mirror, it must return these in a N $\times$ 2 array, where each row contains the x and y coordinates of one of the N vertices. The "orientation" of the array is critical; if it is defined as a 2 $\times$ N array, the model will not work. It is usually easiest to form this output in steps. For example, both `hex2.m` and `lens2.m` define two one-dimensional arrays `x` and `y` containing the x and y coordinates, and then combine them with the command

```
>> reply = [x',y'];
```

where the prime operator transposes the `x` and `y` arrays into column vectors, and `reply` is the variable that is returned by the `genfunc`.

B.2.3 Micromirror Lines (`rqst = 2`*).* The result of requesting this data from the device function should be a M $\times$ 3 array, where each row is a three-element vector `[pt1,pt2,area]` that defines one of the M lines in the image of an individual mirror of the array. In this notation, `pt1` and `pt2` are the indices, in the array of point coordinates described above, of the line's endpoints; and `area` is the index of the area of common height and reflectivity that the line borders. When the line in question is shared by two polygons of different areas, the value of `area` can be set to either one.

B.2.4 Micromirror Seed Points (`rqst = 3`*).* For each polygon in the image that results from drawing the lines of the last paragraph, a seed point needs to be defined. When asked for the seed points, the `genfunc` needs to return a P $\times$ 3 array, where P is the number of seed points. Each row is a three-element vector `[x,y,area]` that defines a seed point, and the index of the area it belongs to.

B.2.5 Physical Extent of Micromirror Image (`rqst = 4`*).* The result of requesting this data item from the `genfunc` is a four-element vector:
`[left,top,right,bottom]`

that contains the coordinates of the corners of the image of an individual micromirror. These should simply be the minimum and maximum values of the x and y coordinates of the vertices.

B.2.6 Reflectivity Table (`rqst = 5`*).* This is a one-dimensional array that contains the reflectivities, ordered by area index number, of the different areas of common height and reflectivity defined above. The reflectivities, defined as the reflected energy divided by the incident energy, should be between 0 (all the energy is absorbed) and 1 (all the energy is reflected).

B.2.7 Height Table (`rqst = 6`*).* This is the same as above, except that the values returned are the heights of the different areas, expressed in the same units of length as the coordinates of the vertices.

B.2.8 Array Lattice Points (`rqst = 7`*).* When asked for the coordinates of the centers of the mirrors in the array, the device function should return a Q $\times$ 2 array, where each row is a two-element vector `[x,y]` containing the x and y coordinates of the center of a micromirror within the array. It is critical that the same unit of length be used for these coordinates as was used for the coordinates of the vertices of an individual mirror. These are used along with the definition of the extent of the image (`rqst = 4`) to convert these coordinates to the pixel coordinates that are used to do the tiling.

B.2.9 Control Surface Vertices (`rqst = 8`*).* The image of the movable surface is constructed in the same way as the image of an individual micromirror. This data item is formatted the same way as the one described for `rqst = 1`.

B.2.10 Control Surface Lines (`rqst = 9`*).* This is similar to the array of line definitions returned for `rqst = 2`. An important difference is that each line of the movable surface image does not need an area index number associated with

it. The movable surface has only two areas, one that moves in response to a control input, and one that doesn't. The difference between the two is defined by the control surface seed points. The output resulting from this request should therefore be a R $\times$ 2 array defining the R lines in the image of an individual movable surface.

B.2.11 Control Surface Seed Points (`rqst = 10`*).* The response to this request should be a S $\times$ 2 array containing the coordinates of the seed points defining the polygons that belong to the movable surface area in the movable surface image. Like the definitions of the lines (`rqst = 9`), area index numbers are not necessary with this data item.

Appendix C. Matlab code listings

C.1 Device functions

C.1.1 hex2.m.

```
function reply = hex2(rqst,spec)

% HEX2 - This function is a genfunc for version 2 of the Peter Roberts
%   Micromirror Modeler. It provides the data for the modeler to create an
%   image of the MUMPS7 hexagonal micromirror array.

% define some constants to make code more readable
Rvert = 1;
Lines = 2;
Rseed = 3;
Rlims = 4;
Reflect = 5;
Heights = 6;
Rlats = 7;
RCvert = 8;
CLines = 9;
RCseed = 10;

% define layer indices:
ring = 4;
gold = 3;
poly2 = 2;
nitride = 1;

% if rvert, rseed, rlims, rcvert or rcseed was requested
if (rqst==Rvert)|(rqst==Rseed)|(rqst==Rlims)|(rqst==RCvert)|(rqst==RCseed)
  % define vertices in real world coordinates: [x,y]
  %   first, compute the coordinates, on axis, of the simple branch:
  xx(1) = 50-2/sin(pi/3);              yy(1) = 0;
  xx(2) = 50;                          yy(2) = 0;
  xx(3) = xx(2)+3.1/sin(pi/3);         yy(3) = 0;
  xx(4) = xx(3)+2/sin(pi/3);           yy(4) = 0;
  xx(5) = xx(4)+3.1/sin(pi/3);         yy(5) = 0;
  xx(6) = xx(5)+2/sin(pi/3);           yy(6) = 0;
  xx(7) = (xx(6)+6/sin(pi/3));         yy(7) = 0;
  %   now make three rotated copies to generate the actual branches:
```

```
for i=1:7
  [x(i),y(i)]       = rot(xx(i),yy(i),pi/4);
  [x(i+21),y(i+21)] = rot(xx(i),yy(i),11*pi/12);
  [x(i+42),y(i+42)] = rot(xx(i),yy(i),19*pi/12);
end
%   next, compute the on axis coordinates of the complex branch:
xx(15) = xx(1);                yy(15) = 0;
xx(16) = xx(4);                yy(16) = 0;
xx(17) = xx(5);                yy(17) = 0;
xx(18) = xx(6);                yy(18) = 0;
xx(19) = xx(7);                yy(19) = 0;
xx( 8) = xx(17)-13*cos(pi/3);  yy( 8) = yy(17)-13*sin(pi/3);
xx( 9) = xx(8)-3.1*cos(pi/6);  yy( 9) = yy(8)+3.1*sin(pi/6);
xx(10) = xx(17)-11*cos(pi/3);  yy(10) = yy(17)-11*sin(pi/3);
xx(11) = xx(10)-5.1*cos(pi/6); yy(11) = yy(10)+5.1*sin(pi/6);
xx(13) = xx(2)-4*cos(pi/3);    yy(13) = yy(2)-4*sin(pi/3);
xx(12) = xx(13)+3.1*cos(pi/6); yy(12) = yy(13)-3.1*sin(pi/6);
xx(14) = xx(16)-4*cos(pi/3);   yy(14) = yy(16)-4*sin(pi/3);
xx(20) = xx(13);               yy(20) = -yy(13);
xx(21) = xx(12);               yy(21) = -yy(12);
%   and make three rotated copies of it:
for i=8:21
  [x(i),y(i)]       = rot(xx(i),yy(i),7*pi/12);
  [x(i+21),y(i+21)] = rot(xx(i),yy(i),15*pi/12);
  [x(i+42),y(i+42)] = rot(xx(i),yy(i),23*pi/12);
end
%   next, define some additional points required to generate an image of the
%   control surface
[x(64),y(64)] = rot(xx(2),yy(2),7*pi/12);
[x(65),y(65)] = rot(xx(2),yy(2),15*pi/12);
[x(66),y(66)] = rot(xx(2),yy(2),23*pi/12);
% now define the output
if rqst == Rvert                                                    % rvert
  reply = [x(1:63)',y(1:63)'];
elseif rqst == Rseed                                                % rseed
  % define seed points in real world coords: [x,y,color]
  reply = [[(x( 1)+x(36))/2,(y( 1)+y(36))/2,gold],    % the mirror
           [(x( 1)+x(13))/2,(y( 1)+y(13))/2,poly2],   % 1st arm
           [(x( 3)+x( 9))/2,(y( 3)+y( 9))/2,poly2],
           [(x( 9)+x(10))/2,(y( 9)+y(10))/2,poly2],
           [(x( 5)+x(18))/2,(y( 5)+y(18))/2,poly2],
           [(x(15)+x(16))/2,(y(15)+y(16))/2,poly2],
```

```
[(x(15)+x(23))/2,(y(15)+y(23))/2,poly2],
[(x(21)+x(25))/2,(y(21)+y(25))/2,poly2],
[(x(17)+x(27))/2,(y(17)+y(27))/2,poly2],
[(x(22)+x(34))/2,(y(22)+y(34))/2,poly2],   % 2nd arm
[(x(24)+x(30))/2,(y(24)+y(30))/2,poly2],
[(x(30)+x(31))/2,(y(30)+y(31))/2,poly2],
[(x(26)+x(39))/2,(y(26)+y(39))/2,poly2],
[(x(36)+x(37))/2,(y(36)+y(37))/2,poly2],
[(x(36)+x(44))/2,(y(36)+y(44))/2,poly2],
[(x(42)+x(46))/2,(y(42)+y(46))/2,poly2],
[(x(38)+x(48))/2,(y(38)+y(48))/2,poly2],
[(x(43)+x(55))/2,(y(43)+y(55))/2,poly2],   % 3rd arm
[(x(45)+x(51))/2,(y(45)+y(51))/2,poly2],
[(x(51)+x(52))/2,(y(51)+y(52))/2,poly2],
[(x(47)+x(60))/2,(y(47)+y(60))/2,poly2],
[(x(57)+x(58))/2,(y(57)+y(58))/2,poly2],
[(x(57)+x( 2))/2,(y(57)+y( 2))/2,poly2],
[(x(63)+x( 4))/2,(y(63)+y( 4))/2,poly2],
[(x(59)+x( 6))/2,(y(59)+y( 6))/2,poly2],
[(x( 3)+x(13))/2,(y( 3)+y(13))/2,nitride], % nitride gaps
[(x(10)+x(12))/2,(y(10)+y(12))/2,nitride],
[(x(10)+x(16))/2,(y(10)+y(16))/2,nitride],
[(x( 5)+x( 9))/2,(y( 5)+y( 9))/2,nitride],
[(x(17)+x(24))/2,(y(17)+y(24))/2,nitride],
[(x(21)+x(23))/2,(y(21)+y(23))/2,nitride],
[(x(24)+x(34))/2,(y(24)+y(34))/2,nitride],
[(x(31)+x(33))/2,(y(31)+y(33))/2,nitride],
[(x(31)+x(37))/2,(y(31)+y(37))/2,nitride],
[(x(26)+x(30))/2,(y(26)+y(30))/2,nitride],
[(x(38)+x(45))/2,(y(38)+y(45))/2,nitride],
[(x(42)+x(44))/2,(y(42)+y(44))/2,nitride],
[(x(45)+x(55))/2,(y(45)+y(55))/2,nitride],
[(x(52)+x(54))/2,(y(52)+y(54))/2,nitride],
[(x(52)+x(58))/2,(y(52)+y(58))/2,nitride],
[(x(47)+x(51))/2,(y(47)+y(51))/2,nitride],
[(x(59)+x( 3))/2,(y(59)+y( 3))/2,nitride],
[(x(63)+x( 2))/2,(y(63)+y( 2))/2,nitride],
[(x( 7)+x(18))/2,(y( 7)+y(18))/2,ring],    % ring
[(x(19)+x(27))/2,(y(19)+y(27))/2,ring],
[(x(28)+x(39))/2,(y(28)+y(39))/2,ring],
[(x(40)+x(48))/2,(y(40)+y(48))/2,ring],
[(x(49)+x(60))/2,(y(49)+y(60))/2,ring],
```

```
            [(x(61)+x( 6))/2,(y(61)+y( 6))/2,ring]];
  elseif rqst == Rlims                                          % rlims
    m = max([max(x),max(y)]);
    reply = [-m,m,m,-m];
  elseif rqst == RCvert                                         % rcvert
    reply = [[x( 2),y( 2)],
             [x(64),y(64)],
             [x(23),y(23)],
             [x(65),y(65)],
             [x(44),y(44)],
             [x(66),y(66)]];
  elseif rqst == RCseed                                         % rcseed
    reply = [0,0];
  end

% if lines was requested
elseif rqst == Lines
% define array of line references:
%   element format: [index of point 1, index of point 2, color]
  reply = [[ 1,15,gold],     % first the mirror itself
           [15,22,gold],
           [22,36,gold],
           [36,43,gold],
           [43,57,gold],
           [57, 1,gold],
           [ 1, 2,poly2],    % then the first flexor arm
           [ 2,13,poly2],
           [13,15,poly2],
           [ 3,11,poly2],
           [11,9 ,poly2],
           [ 9, 4,poly2],
           [ 4, 3,poly2],
           [11,10,poly2],
           [10, 8,poly2],
           [ 5, 8,poly2],
           [10,17,poly2],
           [17,18,poly2],
           [18, 6,ring],
           [ 6, 5,poly2],
           [ 8, 9,poly2],
           [13,12,poly2],
           [12,14,poly2],
```

```
[14,16,poly2],
[16,21,poly2],
[21,20,poly2],
[20,15,poly2],
[20,23,poly2],
[23,22,poly2],
[21,24,poly2],
[24,25,poly2],
[25,16,poly2],
[17,26,poly2],
[26,27,poly2],
[27,18,ring],
[23,34,poly2],    % and the second flexor arm
[34,36,poly2],
[24,32,poly2],
[32,30,poly2],
[30,25,poly2],
[32,31,poly2],
[31,29,poly2],
[26,29,poly2],
[31,38,poly2],
[38,39,poly2],
[39,27,ring],
[29,30,poly2],
[34,33,poly2],
[33,35,poly2],
[35,37,poly2],
[37,42,poly2],
[42,41,poly2],
[41,36,poly2],
[41,44,poly2],
[44,43,poly2],
[42,45,poly2],
[45,46,poly2],
[46,37,poly2],
[38,47,poly2],
[47,48,poly2],
[48,39,ring],
[44,55,poly2],    % and the third flexor arm
[55,57,poly2],
[45,53,poly2],
[53,51,poly2],
```

```
[51,46,poly2],
[53,52,poly2],
[52,50,poly2],
[47,50,poly2],
[52,59,poly2],
[59,60,poly2],
[60,48,ring],
[50,51,poly2],
[55,54,poly2],
[54,56,poly2],
[56,58,poly2],
[58,63,poly2],
[63,62,poly2],
[62,57,poly2],
[62, 2,poly2],
[63, 3,poly2],
[ 4,58,poly2],
[59, 5,poly2],
[ 2, 3,nitride],  % and the nitride gaps
[11,12,nitride],
[10,14,nitride],
[16,17,nitride],
[25,26,nitride],
[23,24,nitride],
[32,33,nitride],
[31,35,nitride],
[37,38,nitride],
[46,47,nitride],
[44,45,nitride],
[53,54,nitride],
[52,56,nitride],
[58,59,nitride],
[ 4, 5,nitride],
[ 6,60,ring],     % and finally the intermirror ring
[ 6, 7,ring],
[ 7,19,ring],
[19,18,ring],
[19,28,ring],
[28,27,ring],
[28,40,ring],
[40,39,ring],
[40,49,ring],
```

```
            [49,48,ring],
            [49,61,ring],
            [61,60,ring],
            [61, 7,ring]];

% if clines was requested
elseif rqst == CLines
% define array of line references:
%   element format: [index of point 1, index of point 2]
  reply = [[1,2],
            [2,3],
            [3,4],
            [4,5],
            [5,6],
            [6,1]];

% if the layer reflectivities were requested
elseif rqst == Reflect
  reply = [0.4,      % nitride
            0.35,     % poly2
            0.9,      % gold
            0.9];     % ring

% if the layer heights were requested
elseif rqst == Heights
  reply = [0,        % nitride
            2.75,     % poly2
            2.75+1.5, % gold
            2.75+1.5];% ring


% if the array lattice points were requested
elseif rqst == Rlats
  numrings = spec;
  % define lattice points in real world coordinates: [x,y]
  b = 100/sqrt(4+1/cos(pi/6));
  width = 2*(b+3.1+2+3.1+5);
  delta1 = width*sin(pi/12);          % define displacement values
  delta2 = width*cos(pi/12);
  delta3 = width*cos(pi/4);
  n=1;
  rlatx(n) = 0; rlaty(n) = 0;         % central mirror
```

```
  for i=1:numrings
    n=n+1;
    rlatx(n) = rlatx(1) + i*delta1;     % 1st mirror of 1st side
    rlaty(n) = rlaty(1) + i*delta2;
    for j=1:i                           % rest of 1st side
      n=n+1;
      rlatx(n) = rlatx(n-1) + delta3;
      rlaty(n) = rlaty(n-1) - delta3;
    end
    for j=1:i                           % 2nd side
      n=n+1;
      rlatx(n) = rlatx(n-1) - delta1;
      rlaty(n) = rlaty(n-1) - delta2;
    end
    for j=1:i                           % 3rd side
      n=n+1;
      rlatx(n) = rlatx(n-1) - delta2;
      rlaty(n) = rlaty(n-1) - delta1;
    end
    for j=1:i                           % 4th side
      n=n+1;
      rlatx(n) = rlatx(n-1) - delta3;
      rlaty(n) = rlaty(n-1) + delta3;
    end
    for j=1:i                           % 5th side
      n=n+1;
      rlatx(n) = rlatx(n-1) + delta1;
      rlaty(n) = rlaty(n-1) + delta2;
    end
    for j=1:i-1                         % 6th side
      n=n+1;
      rlatx(n) = rlatx(n-1) + delta2;
      rlaty(n) = rlaty(n-1) + delta1;
    end
  end
  reply = [rlatx',rlaty'];
end
```

C.1.2 lens2.m.

```
function reply = lens2(rqst,spec)

% LENS2 - This function is a genfunc for version 2 of the Peter Roberts
%   Micromirror Modeler. It provides the data for the modeler to create an
%   image of an orthogonal lenslet array like the ones used with Hartman
%   wavefront sensors.

% define some constants to make code more readable
Rvert = 1;
Lines = 2;
Rseed = 3;
Rlims = 4;
Reflect = 5;
Heights = 6;
Rlats = 7;
RCvert = 8;
CLines = 9;
RCseed = 10;

% define layer indices:
lens = 1;

% if rvert, rseed, rlims, rcvert or rcseed was requested
if (rqst==Rvert)|(rqst==Rseed)|(rqst==Rlims)|(rqst==RCvert)|(rqst==RCseed)
  % define vertices in real world coordinates: [x,y]
  %   first, compute the coordinates, on axis, of the simple branch:
  x(1) = -0.5;              y(1) = -0.5;
  x(2) = -0.5;              y(2) =  0.5;
  x(3) =  0.5;              y(3) =  0.5;
  x(4) =  0.5;              y(4) = -0.5;
  % now define the output
  if rqst == Rvert                                              % rvert
    reply = [x(1:4)',y(1:4)'];
  elseif rqst == Rseed                                          % rseed
    reply = [0,0,lens];
  elseif rqst == Rlims                                          % rlims
    reply = [-0.5,0.5,0.5,-0.5];
  elseif rqst == RCvert                                         % rcvert
    reply = [x(1:4)',y(1:4)'];
  elseif rqst == RCseed                                         % rcseed
    reply = [0,0];
```

```
  end

% if lines was requested
elseif rqst == Lines
% define array of line references:
%   element format: [index of point 1, index of point 2, color]
  reply = [[1,2,lens],
           [2,3,lens],
           [3,4,lens],
           [4,1,lens]];

% if clines was requested
elseif rqst == CLines
% define array of line references:
%   element format: [index of point 1, index of point 2]
  reply = [[1,2],
           [2,3],
           [3,4],
           [4,1]];

% if the layer reflectivities were requested
elseif rqst == Reflect
  reply = [0.9];

% if the layer heights were requested
elseif rqst == Heights
  reply = [0];

% if the array lattice points were requested
elseif rqst == Rlats
  numrows = spec(1);
  numcols = spec(2);
  % define lattice points in real world coordinates: [x,y]
  n = 1;
  for i = 1:numrows
    for j = 1:numcols
      rlatx(n) = i-((numrows+1)/2);
      rlaty(n) = j-((numcols+1)/2);
      n = n+1;
    end
  end
  reply = [rlatx',rlaty'];
```

```
end
```

C.2 Model generating functions

C.2.1 getaxes.m.

```
function [xaxis,uaxis] = getaxes(power,genfunc,spec,table);

% GETAXES - This function returns the real and far field axes of the device
%   described by 'genfunc'.
% usage: [xaxis,uaxis] = getaxes(power,'genfunc',spec,table);
%   'power' defines the number of pixels in the model. This should be the same
%   value that was passed to mama2 to make the array.
%   'genfunc' is a function that defines certain characteristics of the
%   device. genfunc should be written so that genfunc(7,spec) returns a 2xn
%   array containing the real world coordinates of the lattice points in the
%   array of micromirrors.
%   'spec' is an additional argument of genfunc. Here it is
%   used to specify the size of the array, and should be the same value that
%   was given to mama2 to generate the array.
%   'table' is a 2xn array containing the pixel coordinates of the n
%   micromirrors in the array.

% initialize constants:
Rlats = 7;
n = 2^power;

rtable = eval([genfunc '(' int2str(Rlats) ',spec)']);
rx = rtable(:,1);
mx =  table(:,1);
rxmin = min(rx);
rxmax = max(rx);
mxmin = min(mx);
mxmax = max(mx);
tx = 1:n;
ax = (rxmax-rxmin)/(mxmax-mxmin);
bx = rxmax-ax*mxmax;
xaxis = ax*tx+bx;
deltax = xaxis(2)-xaxis(1);
deltau = 1/(deltax*n);
uaxis = -n*deltau/2:deltau:(n/2-1)*deltau;
```

C.2.2 macs2.m.

```
function out = macs2(len,genfunc,spec)

% MACS2 - Make A Control Surface, version 2
%       Constructs an image of the movable surface for an individual
%       micromirror.
%    Usage: mircon = macs(len,'genfunc',spec)
%    The size of the array that will hold the image will be len x len. This
%    should be the same len as used to generate the image of the micromirror
%    with mam2.
%    'genfunc' is another function that defines the type of micromirror to be
%    constructed. It should be written so that:
%      - genfunc(4,spec) returns a four element vector containing the limits,
%        in real world coordinates, of the image:
%            [left,top,right,bottom]
%      - genfunc(8,spec) returns a 2-by-n array containing the real world
%        coordinates of all n vertices in the image of the movable surface:
%            [[x1,y1],
%             [x2,y2],
%             .......,
%             [xn,yn]]
%      - genfunc(9,spec) returns a 2-by-m array that defines all m lines in
%        the image:
%            [[startpt,endpt],
%             ..............]
%        Here 'startpt' and 'endpt' are the indices of the x-coordinates of
%        the points being referred to in the array returned by
%        genfunc(8,spec). The lines should be defined such that they form all
%        convex polygons.
%      - genfunc(10,spec) returns a 2-by-k array that defines all k seed points
%        needed to fill the polygons in the image:
%            [[x1,y1],
%             .......,
%             [xk,yk]]
%      'spec' is not used in any of these components of genfunc. However,
%      genfunc is used by three other functions (mam, mama, and macma) to
%      generate other components of the micromirror array model, and the spec
%      input is used by some of these. For details, type 'help <function>' at
%      the MATLAB prompt.

% define some constants to make the code more readable
Rlims = 4;
```

```
RCvert = 8;
CLines = 9;
RCseed = 10;

% initialize the image array:
mircon = zeros(len);

% get the real world coordinates of the vertices:
rcvert = eval([genfunc '(' int2str(RCvert) ',' int2str(spec) ')']);

% get the array of line references:
clines = eval([genfunc '(' int2str(CLines) ',' int2str(spec) ')']);

% get the real world coordinates of the seed points:
rcseed = eval([genfunc '(' int2str(RCseed) ',' int2str(spec) ')']);

% set up the limits for the array dimensions:
%   matrix limits:
mlims = [1,len,len,1];
%   get the real world limits:
rlims = eval([genfunc '(' int2str(Rlims) ',' int2str(spec) ')']);

% convert rcvert (real world coords) to mcvert (pixel coords):
mcvert = rcvert;
n = size(rcvert,1);
for i=1:n
  [mcvert(i),mcvert(n+i)] = r2m(rcvert(i),rcvert(n+i),rlims,mlims);
end

% convert rcseed to mcseed:
mcseed = rcseed;
n = size(rcseed,1);
for i=1:n
  [mcseed(i),mcseed(n+i)] = r2m(rcseed(i),rcseed(n+i),rlims,mlims);
end

% draw all the lines:
n = size(clines,1);
disp(n)
m = size(mcvert,1);
for i=1:n
  point1 = clines(i);
```

```
  point2 = clines(n+i);
  x1 = mcvert(point1);
  y1 = mcvert(m+point1);
  x2 = mcvert(point2);
  y2 = mcvert(m+point2);
  mircon = pline(x1,y1,x2,y2,1,mircon);
  disp(n-i)
end

% do the fills:
n = size(mcseed,1);
for i=1:n
  x = mcseed(i);
  y = mcseed(n+i);
  mircon = pfilll(x,y,1,mircon,n-i);
%  mircon = setpix(x,y,1,mircon);
end

% all done!
out = mircon;
```

C.2.3 macsa2.m.

```
function out = macsa2(power,element,genfunc,spec)

% MACSA2 - Make A Control Surface Array, version 2
%        Constructs an image of an array of movable micromirror surfaces.
%    Usage: conray = macsas(power,element,'genfunc',spec)
%    The size of the array that will hold the image will be 2^power x 2^power
%    and should be the same as the size of the array containing the image of
%    the mirror array.
%    'element' contains an image of an individual micromirror control surface.
%    'genfunc' is another function that defines the type of micromirror array
%    to be constructed. It can be the same genfunc used by mam to create an
%    individual micromirror, with the following definitions:
%      - genfunc(4,spec) returns a four element vector containing the limits,
%        in real world coordinates, of the image:
%            [left,top,right,bottom]
%        'spec' is not used by this component of genfunc.
%      - genfunc(6,spec) returns a 2xn array containing the real world
%        coordinates of the lattice points of the array:
%            [[x1,y1],
%             [x2,y2],
%             .......,
%             [xn,yn]]
%        macma will place a copy of 'element' at each of these lattice points,
%        centered on the lattice points. Here, 'spec' is used to specify the
%        size, in elements, of the array, and should be the same as that used
%        to generate the image of the array with mama. For example, for a
%        hexagonal array, 'spec' can be an integer denoting the number of
%        rings to be inserted. For an orthogonal array, 'spec' could be a
%        two-element vector specifying the number of rows and columns.
%        For a definition of the other components of genfunc, type 'help mam',
%        'help macm' and 'help mama' from the MATLAB prompt.
%      'spec' is an additional variable for use by genfunc.

% define some constants to make the code more readable
Rlims = 4;
Rlats = 7;

% initialize the image array:
conray = zeros(2^power);

% get the real world coordinates of the lattice points
```

```
rlats = eval([genfunc '(' int2str(Rlats) ',spec)']);

% set up the limits for the array dimensions:
%   matrix limits:
mlims = [1,2^power,2^power,1];
%   real world limits:
erlims = eval([genfunc '(' int2str(Rlims) ',0)']);
w = abs(erlims(3)-erlims(1));
pixsize = w/size(element,1);
m = pixsize * 2^power / 2;
rlims = [-m,m,m,-m];

% convert rvert (real world coords) to mvert (pixel coords):
mlats = rlats;
n = size(rlats,1);
for i=1:n;
  [mlats(i),mlats(n+i)] = r2m(rlats(i),rlats(n+i),rlims,mlims);
end

% insert all the control surfaces:
offset = size(element,1)/2;
n = size(mlats,1);
for i=1:n
  conray = insert(i*element,conray,mlats(i)-offset,mlats(n+i)-offset);
  disp(n-i)
end

out = conray;
```

C.2.4 malut2.m.

```
function out = malut2(power,element,genfunc,spec)

% MALUT2 - Make A Look-Up Table, version 2
%        Constructs a look-up table for controlling an array of micromirrors.
%    Usage: table = malut(power,element,'genfunc',spec)
%    The arguments to this function should be exactly the same as those given
%    mama to generate the micromirror array. The result will be a 2-by-n
%    matrix containing the pixel coordinates of the centers of each mirror in
%    the array. This table can be used by other functions to modify the array
%    to simulate the application of a control input. When an image of control
%    surfaces is generated by macma, each individual surface will contain the
%    value of the index of that surface's center point in the table generated
%    by malut.

% define some constants to make the code more readable
Rlims = 4;
Rlats = 7;

% get the real world coordinates of the lattice points
rlats = eval([genfunc '(' int2str(Rlats) ',spec)']);

% set up the limits for the array dimensions:
%   matrix limits:
mlims = [1,2^power,2^power,1];
%   real world limits:
erlims = eval([genfunc '(' int2str(Rlims) ',0)']);
w = abs(erlims(3)-erlims(1));
pixsize = w/size(element,1);
m = pixsize * 2^power / 2;
rlims = [-m,m,m,-m];

% convert rvert (real world coords) to mvert (pixel coords):
mlats = rlats;
n = size(rlats,1);
for i=1:n;
  [mlats(i),mlats(n+i)] = r2m(rlats(i),rlats(n+i),rlims,mlims);
end

% all done!
out = mlats;
```

C.2.5 mam2.m.

```
function out = mam2(len,genfunc,spec)

% MAM2 - Make A Mirror, version 2
%       Constructs an image of an individual micromirror.
%    Usage: mirror = mam2(len,'genfunc',spec)
%    The size of the array that will hold the image will be len x len.
%    'genfunc' is another function that defines the type of micromirror to be
%    constructed. It should be written so that:
%      - genfunc(1,spec) returns a 2-by-n array containing the real world
%        coordinates of all vertices in the image:
%            [[x1,y1],
%             [x2,y2],
%             .......,
%             [xn,yn]]
%      - genfunc(2,spec) returns a 3-by-m array that defines all the lines in
%        the image:
%            [[startpt,endpt,index],
%             ....................]
%        Here 'startpt' and 'endpt' are the indices of the x-coordinates of the
%        points being referred to in the array returned by genfunc(1,spec).
%        The lines should be defined such that they form all convex polygons.
%      - genfunc(3,spec) returns a 3-by-k array that defines all the seed
%        points needed to fill the polygons in the image:
%            [[x1,y1,index],
%             ............,
%             [xk,yk,index]]
%      - genfunc(4,spec) returns a four element vector containing the limits,
%        in real world coordinates, of the image:
%            [left,top,right,bottom]
%      'spec' is an additional argument to genfunc that depends on the
%      particular genfunc being used.

% define some constants
Rvert = 1;
Lines = 2;
Rseed = 3;
Rlims = 4;

% initialize the image array:
mirror = zeros(len);
```

```
% get the real world coordinates of the vertices:
rvert = eval([genfunc '(' int2str(Rvert) ',' int2str(spec) ')']);

% get the array of line references:
lines = eval([genfunc '(' int2str(Lines) ',' int2str(spec) ')']);

% get the real world coordinates of the seed points:
rseed = eval([genfunc '(' int2str(Rseed) ',' int2str(spec) ')']);

% set up the limits for the array dimensions:
%   matrix limits:
mlims = [1,len,len,1];
%   get the real world limits:
rlims = eval([genfunc '(' int2str(Rlims) ',' int2str(spec) ')']);

% convert rvert (real world coords) to mvert (pixel coords):
mvert = rvert;
n = size(rvert,1);
for i=1:n
  [mvert(i),mvert(n+i)] = r2m(rvert(i),rvert(n+i),rlims,mlims);
end

% convert rseed to mseed:
mseed = rseed;
n = size(rseed,1);
for i=1:n
  [mseed(i),mseed(n+i)] = r2m(rseed(i),rseed(n+i),rlims,mlims);
end

% draw all the lines:
n = size(lines,1);
disp(n)
m = size(mvert,1);
for i=1:n
  point1 = lines(i);
  point2 = lines(n+i);
  color  = lines(2*n+i);
  x1 = mvert(point1);
  y1 = mvert(m+point1);
  x2 = mvert(point2);
  y2 = mvert(m+point2);
  mirror = pline(x1,y1,x2,y2,color,mirror);
```

```
  disp(n-i)
end

% do the fills:
n = size(mseed,1);
for i=1:n
  x = mseed(i);
  y = mseed(n+i);
  color = mseed(2*n+i);
  mirror = pfilll(x,y,color,mirror,n-i);
%  mirror = setpix(x,y,color,mirror);
end

% all done!
out = mirror;
```

C.2.6 mama2.m.

```
function out = mama2(power,element,genfunc,spec)

% MAMA2 - Make A Mirror Array, version 2
%        Constructs an image of an array of micromirrors.
%    Usage: marray = mama(power,element,'genfunc',spec)
%    The size of the array that will hold the image will be 2^power x 2^power.
%    'element' contains an image of an individual micromirror.
%    'genfunc' is another function that defines the type of micromirror array
%    to be constructed. It can be the same genfunc used by mam to create an
%    individual micromirror, with the following definitions:
%      - genfunc(4,spec) returns a four element vector containing the limits,
%        in real world coordinates, of the image:
%            [left,top,right,bottom]
%        'spec' is not used by this component of genfunc
%      - genfunc(7,spec) returns a 2xn array containing the real world
%        coordinates of the lattice points of the array:
%            [[x1,y1],
%             [x2,y2],
%             .......,
%             [xn,yn]]
%        mama will place a copy of 'element' at each of these lattice points,
%        centered on the lattice points. Here, 'spec' is used to specify the
%        size, in elements, of the array. For example, for a hexagonal array,
%        'spec' can be an integer denoting the number of rings to be inserted.
%        For an orthogonal array, 'spec' could be a two-element vector
%        specifying the number of rows and columns.
%        For a definition of the other components of genfunc, type 'help mam'
%        from the MATLAB prompt.
%      'spec' is an additional variable for use by genfunc.

% define some constants to make the code more readable
Rlims = 4;
Rlats = 7;

% initialize the image array:
marray = zeros(2^power);

% get the real world coordinates of the lattice points
rlats = eval([genfunc '(' int2str(Rlats) ',spec)']);

% set up the limits for the array dimensions:
```

```
%   matrix limits:
mlims = [1,2^power,2^power,1];
%   real world limits:
erlims = eval([genfunc '(' int2str(Rlims) ',0)']);
w = abs(erlims(3)-erlims(1));
pixsize = w/size(element,1);
m = pixsize * 2^power / 2;
rlims = [-m,m,m,-m];

% convert rvert (real world coords) to mvert (pixel coords):
mlats = rlats;
n = size(rlats,1);
for i=1:n;
  [mlats(i),mlats(n+i)] = r2m(rlats(i),rlats(n+i),rlims,mlims);
end

% insert all the micromirrors:
offset = size(element,1)/2;
n = size(mlats,1);
for i=1:n
  marray = insert(element,marray,mlats(i)-offset,mlats(n+i)-offset);
  disp(n-i)
end

% do some touching up
border = max(max(element));
for i=1:2^power
  for j=1:2^power
    color = marray(i,j);
    if (color>border)
      marray(i,j) = border;
    end
  end
end

% all done!
out = marray;
```

C.2.7 mhm2.m.

```
function hiray = mhm2(indexray,genfunc,spec);

% MHM2 - Make Height Matrix, version 2
%   Constructs an image matrix of height values.
%   usage: hiray = mhm2(indexray,'genfunc',spec)
%   'indexray' is an image matrix where each pixel contains an index showing
%   what layer of the MEMS device that pixel belongs to.
%   'genfunc' is another function that defines the type of micromirror to be
%   constructed. Presumably, it will also have been used to generate
%   'indexray'.
%   It should be written so that:
%     - genfunc(6,spec) returns a 1D array containing the height, in microns,
%       of the various layers of the device being modeled. The index of each
%       value within the array should match the indices contained in
%       'indexray'. 'spec' can be used to specify an additional variable for
%       genfunc, if necessary.

% define some constants to make the code more readable
Heights = 6;

% initialize the output
hiray = indexray;

% and do some replacing
hite = eval([genfunc '(' int2str(Heights) ',' int2str(spec) ')']);
n = length(hite);
disp(n)
for i=1:n
  hiray = replace(i,hite(i),hiray);
  disp(n-i)
end
```

C.2.8 mrm2.m.

```
function refray = mrm2(indexray,genfunc,spec);

% MRM2 - Make Reflectivity Matrix, version 2
%   Constructs an image matrix of reflectivity values.
%   usage: refray = mrm2(indexray,'genfunc',spec)
%   'indexray' is an image matrix where each pixel contains an index showing
%   what layer of the MEMS device that pixel belongs to.
%   'genfunc' is another function that defines the type of micromirror to be
%   constructed. Presumably, it will also have been used to generate
%   'indexray'.
%   It should be written so that:
%     - genfunc(5,spec) returns a 1D array containing the reflectivity values
%       of the various layers of the device being modeled. The index of each
%       value within the array should match the indices contained in
%       'indexray'. 'spec' can be used to specify an additional variable. For
%       example, if genfunc is written so that the reflectivity values are
%       dependent on wavelength, 'spec' can be used to pass the wavelength.

% define some constants to make the code more readable
Reflect = 5;

% initialize the output
refray = indexray;

% and do some replacing
ref = eval([genfunc '(' int2str(Reflect) ',' int2str(spec) ')']);
n = length(ref);
disp(n)
for i=1:n
  refray = replace(i,ref(i),refray);
  disp(n-i)
end
```

C.3 Model controlling functions

C.3.1 abcorr.m.

```
function out = abcorr(wavefront,wavelength,hiray,conray,respray,table)

% ABCORR - Deflects an array of micromirrors to compensate for an aberrated
%   wavefront.
%   usage: defmarray = abcorr(wavefront,wavelength,hiray,conray,respray,table)
%   'wavefront' is a real nxn matrix that contains the phase of a wavefront.
%   The magnitude of each pixel represents the distance by which the phase of
%   the wavefront at that pixel is advanced with respect to a reference plane.
%   'wavelength' is simply the wavelength of the incoming wavefront.
%   'hiray' is the array containing the imae of the heights of the
%   individual pizels in the micromirror array.
%   'conray' is the array containing the image of the indexed control
%   surfaces of the micromirror array.
%   'respray' is the array contining an image of the response to control
%   inputs for each pixel in the array.
%   'table' is a n-by-2 lookup table containing the pixel coordinates of the
%   n micromirrors in the array.

out = hiray;
n = size(hiray,1);
for j=1:n
  for k=1:n
    if conray(j,k) ~= 0
      x = table(conray(j,k));
      y = table(conray(j,k)+size(table,1));
      deflection = wavefront(x,y)/2;
      phi = angle(exp(i*2*pi*deflection/wavelength));
      dz = wavelength*phi/(2*pi);
      if dz<0
        dz = dz + wavelength;
      end
      out(j,k) = hiray(j,k) - (dz*respray(j,k));
    end
  end
  disp(n-j)
end
```

C.3.2 parabolize2.m.

```
function out = parabolize2(deflection,lambda,hiray,conray,respray,table)

% PARABOLIZE2 - Adds a parabolic correction to an array of micromirrors.
%   usage: defmarray = parabolize2(deflection,lambda,hiray,conray,respray,table)
%     'deflection' is the total amount of deflection desired at the edges, in
%     wavelengths.
%     'lambda' is the wavelength for which diffraction is to be modeled. It
%     must be in the same units as 'hiray'.
%     'hiray' is the array containing the image of the heights of the
%     individual pixels in the micromirror array.
%     'conray' is the array containing the image of the indexed control
%     surfaces of the micromirror array.
%     'respray' is the array containing the image of the response to control
%     inputs for each pixel in the array.
%     'table' is a n-by-2 lookup table containing the pixel coordinates of the
%     n micromirrors in the array.

out = hiray;
n = size(hiray,1);
xray = table(:,1)-size(hiray,1)/2;
yray = table(:,2)-size(hiray,2)/2;
r2ray = (xray .* xray) + (yray .* yray);
r2max = max(r2ray);
phimax = 2*pi*deflection;
for j=1:n
  for k=1:size(hiray,2)
    if conray(j,k) ~= 0
      phi = phimax*(1-r2ray(conray(j,k))/r2max);
      phi = angle(exp(i*phi));
      dz = lambda*phi/(2*pi);
      if dz<0
        dz = dz + lambda;
      end
      out(j,k) = hiray(j,k) - (dz*respray(j,k));
    end
  end
  disp(n-j)
end
```

C.3.3 piston2.m.

```
function out = piston2(deflection,lambda,hiray,conray,respray,table)

% PISTON2 - Deflects avery mirror in an array of micromirrors by the same
%   amount.
%   usage: defmarray = piston2(deflection,lambda,hiray,conray,respray,table)
%     'deflection' is the total amount of deflection desired, in wavelengths.
%     'lambda' is the wavelength for which diffraction is to be modeled. It
%     must be in the same units as 'hiray'.
%     'hiray' is the array containing the image of the heights of the
%     individual pixels in the micromirror array.
%     'conray' is the array containing the image of the indexed control
%     surfaces of the micromirror array.
%     'respray' is the array containing the image of the response to control
%     inputs for each pixel in the array.
%     'table' is a n-by-2 lookup table containing the pixel coordinates of the
%     n micromirrors in the array.

out = hiray;
n = size(hiray,1);
phi = 2*pi*deflection;
dz = lambda*phi/(2*pi);
if dz<0
  dz = dz + lambda;
end
for j=1:n
  for k=1:n
    if conray(j,k) ~= 0
      out(j,k) = hiray(j,k) - (dz*respray(j,k));
    end
  end
  disp(n-j)
end
```

C.3.4 push2.m.

```
function out = push2(deflection,lambda,hiray,conray,respray,table,index)

% PUSH2 - Adds a phase component to an individual micromirror in an array
%         to simulate a deflection of that micromirror.
%   usage: defmarray = push2(deflection,lambda,hiray,conray,respray,table,index)
%     'deflection' is the total amount of deflection desired, in wavelengths.
%     'lambda' is the wavelength of the light for which the diffraction
%     pattern is to be calculated.
%     'hiray' is the array containing the image of the heights of the
%     individual pixels in the micromirror array.
%     'conray' is the array containing the image of the indexed control
%     surfaces of the micromirror array.
%     'respray' is the array containing the image of the response to control
%     inputs for each pixel in the array.
%     'table' is a 2-by-n lookup table containing the pixel coordinates of the
%     n micromirrors in the array.
%     'index' is the index of the individual micromirror to be deflected.

out = hiray;
i = sqrt(-1);
n = size(hiray,1);
m = size(hiray,2);
phi = angle(exp(i*2*pi*deflection));
dz = lambda*phi/(2*pi);
if dz<0
  dz = dz + lambda;
end
for j=1:n
  for k=1:m
    if conray(j,k) == index
      out(j,k) = hiray(j,k) - (dz*respray(j,k));
    end
  end
  disp(n-j)
end
```

C.3.5 tilt2.m.

```
function out = tilt2(deflection,lambda,hiray,conray,respray,table)

% TILT2 - Adds a tilt correction to an array of micromirrors.
%   usage: tiltray = tilt2(deflection,lambda,hiray,conray,respray,table)
%     'deflection' is the total amount of deflection desired at the edges, in
%     wavelengths.
%     'lambda' is the wavelength for which diffraction is to be modeled. It
%     must be in the same units as 'hiray'.
%     'hiray' is the array containing the image of the heights of the
%     individual pixels in the micromirror array.
%     'conray' is the array containing the image of the indexed control
%     surfaces of the micromirror array.
%     'respray' is the array containing the image of the response to control
%     inputs for each pixel in the array.
%     'table' is a 2-by-n lookup table containing the pixel coordinates of the
%     n micromirrors in the array.

out = hiray;
n = size(hiray,1);
xmin = min(table(:,1));
xmax = max(table(:,1));
phimax = 2*pi*deflection;
for j=1:n
  for k=1:size(hiray,2)
    if conray(j,k) ~= 0
      phi = phimax*(table(conray(j,k))-xmin)/(xmax-xmin);
      phi = angle(exp(i*phi));
      dz = lambda*phi/(2*pi);
      if dz<0
        dz = dz + lambda;
      end
      out(j,k) = hiray(j,k) - (dz*respray(j,k));
    end
  end
  disp(n-j)
end
```

C.4 Diffraction simulation functions

C.4.1 diffract.m.

```
function diff = diffract(wavefront,wavelength,refray,hiray);

% DIFFRACT - This function computes the diffraction pattern caused by
%   reflecting a given wavefront off a device with the given reflectivity
%   and height patterns.
% usage: diff = diffract(wavefront,wavelength,refray,hiray);
%   'wavefront' is a real nxn matrix that contains the phase of a wavefront.
%   The magnitude of each pixel represents the distance by which the phase of
%   the wavefront at that pixel is advanced with respect to a reference plane.
%   'wavelength' is simply the wavelength of the incoming wavefront.
%   'refray' is a real nxn matrix that contains the reflectivity image of the
%   device being modeled.
%   'hiray' is a real nxn matrix that contains the height image of the device
%   being modeled.
%   'diff' is the resulting field strength of the far field diffraction
%   pattern. Its magnitude squared is the intensity of the pattern.
% Note: All three matrix arguments must be the same size. Also, this routine
%   uses MATLAB's FFT function, which works best when n is an integer power
%   of 2.

if (size(wavefront)==size(refray))&(size(wavefront)==size(hiray))
  device = refray .* exp(i*2*pi*(2*hiray)/wavelength);
  diff = fftshift(fft2(fftshift(device .* exp(i*2*pi*wavefront/wavelength))));
else
  diff = 'DIFFRACT error: Matrix arguments must be the same size';
end
```

C.4.2 masw.m.

```
function out = masw(deflection,deltax,power);

% MASW - Make A Spherical Wavefront
% usage: wavefront = masw(deflection,deltax,power);
%   'wavefront' is a real nxn matrix that contains the phase of a wavefront.
%   The magnitude of each pixel represents the amount, in the same units as
%   deltax, by which the phase of the wavefront at that pixel is advanced with
%   respect to a reference plane.
%   'deflection' is the amount by which the phase at the center of the image
%   will be advanced. The corners of the image will be used to define the
%   phase reference plane.
%   'deltax' is the desired pixel size. This should be the same as that of the
%   device image that the wavefront is intended to be bounced off of, and can
%   be found using the getaxes function.
%   'power' defines the size of the output matrix: 2^power x 2^power.

% set up some constants
n = 2^power;
center = (n/2)+1;

% calculate r2 for each pixel
for j=1:n
  for k=1:n
    r2(j,k) = ((j-center)^2 + (k-center)^2) * deltax^2;
  end
  disp(2*n-j)
end

% calculate the deflection as a function of r2
d = deflection;
bigr = (d^2 + max(max(r2))) / (2*d)
for j=1:n
  for k=1:n
    out(j,k) = d - bigr + sqrt(bigr^2-r2(j,k));
  end
  disp(n-j)
end
```

C.4.3 radavgp.m.

```
function radval=radavgp(psfold)

% FUNCTION RADAVG
% Calculates the radial average of an input array and returns a vector
% with the radially averaged output. Only works on square arrays.
%
% obtained from Dr. Roggemann
% slightly modified by Peter Roberts
%
% usage: radval=radavgp(psfold)

if size(psfold,1)~=size(psfold,2)
  radval = 'Argument must be a square matrix.';

else
  imagesize = size(psfold,1);
  accu=zeros(1,(imagesize/2)+2);
  counter=zeros(1,(imagesize/2)+2);
  cen=(imagesize/2)+1;
  for r=1:imagesize,
    for c=1:imagesize,
      dx=c-cen;
      dy=r-cen;
      rad=sqrt(dx^2+dy^2);
      q=round(rad)+1;
      if q<=cen+1,
        accu(q)=accu(q)+psfold(r,c);
        counter(q)=counter(q)+1;
      end;
    end;
  end;
end;

radval=accu./counter;
```

C.4.4 show.m.

```
function show(data)
% SHOW(data) displays a grayscale image of the input array 'data'.

ddata = 255*(data/max(max(data)));
image(ddata);
colormap(gray(256));
axis('image');
axis('xy');
```

C.4.5 show2.m.

```
function show2(x,y,data)
% SHOW2(x,y,data) displays a grayscale image of the input array 'data'.

ddata = 255*(data/max(max(data)));
image(x,y,ddata);
colormap(gray(256));
axis('image');
axis('xy');
```

C.4.6 showdif.m.

```
function intensity = showdif(diff,wavelength,device,genfunc,spec,table);

% SHOWDIF - This function displays the log of the intensity of a diffraction
%   pattern caused by reflecting a given wavefront off a device described by
%   'genfunc'.
% usage: intensity = showdif(diff,wavelength,device,'genfunc',spec,table);
%   'diff' is the field strength of the far field diffraction pattern. Its
%   magnitude squared is the intensity of the pattern, which will be
%   displayed in the bottom half of the figure.
%   'wavelength' is simply the wavelength of the incoming wavefront.
%   'device' is a real nxn matrix that contains an image of the device being
%   modeled. This image will be displayed in the top half of the figure.
%   'genfunc' is a function that defines certain characteristics of the
%   device. showdif uses this to correctly display the axes in the figure.
%   genfunc should be written so that genfunc(7,spec) returns a 2xn array
%   containing the real world coordinates of the lattice points in the array
%   of micromirrors.
%   'spec' is an additional argument of genfunc. Here it is
%   used to specify the size of the array, and should be the same value that
%   was given to mama2 to generate the array.
%   'table' is a 2xn array containing the pixel coordinates of the n
%   micromirrors in the array.

% initialize constants:
Rlats = 7;

% top half of the figure:
subplot(211);
rtable = eval([genfunc '(' int2str(Rlats) ',' int2str(spec) ')']);
rx = rtable(:,1); ry = rtable(:,2);
mx =  table(:,1); my =  table(:,2);
rxmin = min(rx); rymin = min(ry);
rxmax = max(rx); rymax = max(ry);
mxmin = min(mx); mymin = min(my);
mxmax = max(mx); mymax = max(my);
tx = 1:size(device,1);
bx = (rxmax-rxmin)/(mxmax-mxmin);
ax = (rxmax-bx)/mxmax;
x = ax*tx+bx;
deltax = x(2)-x(1);
ty = 1:size(device,2);
```

```
by = (rymax-rymin)/(mymax-mymin);
ay = (rymax-by)/mymax;
y = ay*ty+by;
deltay = y(2)-y(1);
show2(x,y,device);

% bottom half of the figure:
subplot(212);
intensity = abs(diff).^2;
deltau = 1/(deltax*size(device,1));
u = -size(device,1)*deltau/2:deltau:(size(device,1)/2-1)*deltau;
deltav = 1/(deltay*size(device,2));
v = -size(device,2)*deltav/2:deltav:(size(device,2)/2-1)*deltav;
show2(u,v,log(intensity));

% done!
subplot;
```

C.5 Low-level functions

C.5.1 insert.m.

```
function out = insert(in1,in2,x,y)

% INSERT(in1,in2,x,y) - Inserts the matrix 'in1' into the larger matrix 'in2'.
%                       The (1,1) point of 'in1' gets mapped to (x,y) in 'in2'.

if (size(in1,1)>size(in2,1))|(size(in1,2)>size(in2,2))
  out = 'Incompatible matrix sizes'
else
  for i=1:min(size(in1,1),size(in2,1)-x)
    if (x+i-1)>=1
      for j=1:min(size(in1,2),size(in2,2)-y)
        if (y+j-1)>=1
          in2(x+i-1,y+j-1) = in2(x+i-1,y+j-1) + in1(i,j);
        end
      end
    end
  end
  out = in2;
end
```

C.5.2 pfilll.m.

```
function out = pfilll(xin,yin,color,image,n)

% PFILLL(X,Y,COLOR,PICTURE,N) fills the area around (x,y) with 'color'.
%  WARNING: This works properly only with fully convex areas.
%  This is a special version of pfill for use with mahm2 only.

                                        % temporarily add a one pixel border
image = [99*ones(size(image,1),1),image,99*ones(size(image,1),1)];
image = [99*ones(size(image,2),1),image',99*ones(size(image,2),1)]';
window = image;
x = round(xin+1);                       % x and y are the current coords
y = round(yin+1);
oldcolor = window(x,y);
while image(x+1,y) == oldcolor,         % find the right end of the current
  x = x+1;                              %  row
end
notdone = 1;
while notdone,                          % find the top left pixel in the fill
  if image(x,y+1) == oldcolor,          %  area
    while image(x,y+1) == oldcolor,
      y = y+1;
    end
    while image(x+1,y) == oldcolor,
      x = x+1;
    end
  elseif image(x-1,y) == oldcolor,
    x = x-1;
  else notdone = 0;
  end
end

notdone = 1;                            % start setting pixels
while notdone,
  if image(x+1,y) == oldcolor,
    notdone = 0;
    if image(x,y-1) == oldcolor,
      nextx = x;
      nexty = y-1;
      notdone = 1;
    end
    while image(x+1,y) == oldcolor,
```

```
      window = setpix(x,y,color,window);    % fill the current line
      if image(x,y-1) == oldcolor,          % while looking to see if the next
        nextx = x;                          %   line needs to be filled
        nexty = y-1;
        notdone = 1;
      end
      x = x+1;
    end
    if image(x,y-1) == oldcolor,
      nextx = x;
      nexty = y-1;
      notdone = 1;
    end
    window = setpix(x,y,color,window);
  elseif image(x,y-1) == oldcolor,
    window = setpix(x,y,color,window);
    nextx = x;
    nexty = y-1;
    notdone = 1;
  else
    window = setpix(x,y,color,window);
    notdone = 0;
  end
  if notdone,                           % if another line is needed
    x = nextx;
    y = nexty;
    disp([n y])
    while image(x-1,y) == oldcolor,     % find its leftmost point
      x = x-1;
    end
  end
end
out = window(2:size(window,1)-1,2:size(window,2)-1);
```

C.5.3 pline.m.

```
function out = pline(x1,y1,x2,y2,color,image)

% PLINE(X1,Y1,X2,Y2,COLOR,IMAGE) draws a line from (x1,y1) to (x2,y2).
%   Note: PLINE stands for Pete's LINE function. Matlab already has a function
%   called line that does something else.

[xmax,ymax] = size(image);
image = setpix(x1,y1,color,image);
image = setpix(x2,y2,color,image);
if (x1==x2)&(x1>=1)&(x1<=xmax)              % if the line is in the x-direction:
  for y=min(y1,y2):max(y1,y2),                % define range to scan y
    if (y>=1)&(y<=ymax)                       % boundary check
      image = setpix(x1,y,color,image);       % set the next pixel
    end
  end
elseif (y1==y2)&(y1>=1)&(y1<=ymax)          % if the line is in the y-direction:
  for x=min(x1,x2):max(x1,x2),                % define range to scan x
    if (x>=1)&(x<=xmax)                       % boundary check
      image = setpix(x,y1,color,image);       % set the next pixel
    end
  end
elseif abs(y2-y1)<=abs(x2-x1)               % if the |slope| <= 1:
  if x1<x2                                    % find the left most point
    sx=x1; sy=y1; fx=x2; fy=y2;                 % set the start and finish points
  else
    sx=x2; sy=y2; fx=x1; fy=y1;
  end
  dy = (fy-sy)/(fx-sx);                       % calculate the y increment
  y = sy;                                     % initialize y
  for x=sx:fx,                                % define range to scan x
    if (x>0)&(x<=xmax)&(y>0)&(y<=ymax)
      image = setpix(x,y,color,image);        % set the next pixel
    end
    y = sy+(x+1-sx)*dy;                       % compute the next y
  end
else                                        % if the |slope| > 1:
  if y1<y2                                    % find the lower point
    sx=x1; sy=y1; fx=x2; fy=y2;                 % set the start and finish points
  else
    sx=x2; sy=y2; fx=x1; fy=y1;
  end
```

```
  dx = (fx-sx)/(fy-sy);                    % calculate the x increment
  x = sx;                                  % initialize x
  for y=sy:fy,                             % define range to scan y
    if (x>0)&(x<=xmax)&(y>0)&(y<=ymax)
      image = setpix(x,y,color,image);     % set the next pixel
    end
    x = sx+(y+1-sy)*dx;                    % compute the next x
  end
end
out = image;                             % all done!
```

C.5.4 r2m.m.

```
function [mx,my] = r2m(rx,ry,rlims,mlims)

% R2M - Converts the real world coordinates (rx,ry) to matrix coordinates
%    (mx,my). 'rwlims' and 'mlims' contain the limits of the real world
%    and matrix representations that are to be mapped to each other in
%    a standard [left, top, right, bottom] format.

ml = mlims(1); mt = mlims(2); mr = mlims(3); mb = mlims(4);
rl = rlims(1); rt = rlims(2); rr = rlims(3); rb = rlims(4);
my = (mr-ml)*(rx-rl)/(rr-rl) + ml;
mx = (mt-mb)*(ry-rb)/(rt-rb) + mb;
```

C.5.5 replace.m.

```
function out = replace(old, new, in);

% usage: out = replace(old,new,in)

out = in;
for i=1:size(in,1)
  for j=1:size(in,2)
    if in(i,j)==old
      out(i,j)=new;
    end
  end
end
```

C.5.6 rot.m.

```
function [xout,yout] = rot(x,y,alpha)

% ROT - Rotates the point (x,y) by alpha radians about the origin.
%    Usage: [xout,yout] = rot(x,y,alpha)

xout = x*cos(alpha) - y*sin(alpha);
yout = x*sin(alpha) + y*cos(alpha);
```

C.5.7 setpix.m.

```
function out = setpix(x,y,color,image)

% SETPIX(X,Y,COLOR,IMAGE) sets in(x,y) to 'color'.

[xmax,ymax] = size(image);
if (x>=1)&(x<=xmax)&(y>=1)&(y<=ymax)
  image(x,y) = color;
end
out = image;
```

Bibliography

1. Bell, Trudy E. "Electronics and the stars," *IEEE Spectrum*, 16–24 (August 1995).
2. Comtois, John H. *Structures and techniques for implementing and packaging complex, large scale microelectromechanical systems using foundry fabrication processes*, PhD dissertation, AFIT/DS/ENG/96-04, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, 1996.
3. Couch, Leon W. II. *Digital and Analog Communication Systems.* New York: Macmillan Publishing Co., 1983.
4. Ealey, Mark A. and John F. Washeba. "Continuous facesheet low voltage deformable mirrors," *Optical Engineering, 29*:1191–1198 (October 1990).
5. Fried, David L. "Statistics of a geometric representation of wavefront distortion," *Journal of the Optical Society of America, 55*:1427–1435 (November 1965).
6. Fried, David L. "Optical resolution through a randomly inhomogeneous medium for very long and very short exposures," *Journal of the Optical Society of America, 56*:1372–1379 (October 1966).
7. Gardner, Chester S. and Byron M. Welsh. "Perfomrance analysis of adaptive-optics systems using laser guide stars and slope sensors," *Journal of the Optical Society of America A, 6*:1913–1923 (December 1989).
8. Gardner, Chester S., et al., "Design and performance analysis of adaptive optical telescopes using laser guide stars," *Proceedings of the IEEE, 78*:1721–1743 (November 1990).
9. Gleick, James. *Chaos.* New York: Penguin Books, 1987.
10. Gonzalez, Rafael C. and Richard E. Woods. *Digital Image Processing.* New York: Addison-Wesley, 1992.
11. Goodman, Joseph W. *Introduction to Fourier Optics, 2nd Edition.* New York: McGraw-Hill, 1996.
12. Hardy, John W. "Active optics: A new technology for the control of light," *Proceedings of the IEEE, 66*:651–697 (1978).
13. Hardy, John W. "Adaptive Optics," *Scientific American*, 60 (June 1994).
14. Hurlburd, Bill and David Sandler. "Segmented mirrors for atmospheric compensation," *Optical Engineering, 29*:1186–1190 (October 1990).
15. Hick, Shaun R. *Demonstrating Optical Aberration Correction with a MEMS micro-mirror device*, MS thesis, AFIT/GAP/ENG/96D-7, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, 1996.

16. Lachenbruch, David. "Sony's 'Micromirror' Projector," *Electronics Now*, 6 (September 1995).

17. Michalicek, M. Adrian. *Design, fabrication, modeling, and testing of surface-micromachined micromirror devices*, MS thesis, AFIT/GE/ENG/95J-01, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, 1995.

18. Newton, Isaac. *Opticks*. New York: Dover Publications, 1952.

19. Nisenson, Peter and Richard Barakat. "Partial atomspheric correction with adaptive optics," *Journal of the Optical Society of America A*, *4*:2249–2253 (December 1987).

20. Roggemann, Michael C. "Limited degree-of-freedom adaptive optics and image reconstruction," *Applied Optics*, *30*:4227–4233 (10 October 1991).

21. Roggemann, Michael C. "Optical perfomance of fully and partially compensated adaptive optics systems using least-squares and minimum variance phase reconstructors," *Computers Elect. Engng, Vol. 18, No. 6*:451–466 (1992).

22. Sallberg, Scott A. *Maximum Likelihood Estimation of Wave Front Slopes using a Hartmann-type Sensor*. MS thesis, AFIT/GE/ENG/95D-23, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, 1995.

23. Thompson, Laird A. "Adaptive optics in astronomy," *Physics Today*, 24–31 (December 1994).

24. Tyson, Robert K. "Adaptive optics system performance approximations for atmospheric turbulence correction," *Optical Engineering*, *29*:1165–1173 (October 1990).

Vita

Peter C. Roberts graduated from Lycée Pierre de Fermat in Toulouse, France in June 1985. Within a month, he was a cadet at the U. S. Air Force Academy in Colorado Springs, CO, where he studied Space Physics, earning a Bachelor of Science degree in May 1989. His first assignment was with the 3246th Test Wing, Eglin AFB, FL, as an electro-optical test engineer. While at Eglin, he also attended the University of West Florida's graduate business school, and received a Master of Business Administration degree in December 1992. In May 1993, he was reassigned to the Air Force Operational Test and Evaluation Center, Kirtland AFB, NM, as a C^4I systems operations analyst. Two years later, he became a student at the Air Force Institute of Technology, where he studied Physics and graduated in December 1996 with a Master of Science degree. His next assignment was with the National Air Intelligence Center, Wright-Patterson AFB, OH.

Permanent address: 25 Cherry Street
Springboro, OH 45066